Algorithms in the Nashlib set in various programming languages

John C Nash, retired professor, University of Ottawa   Peter Olsen, retired ??

11/01/2021

Contents

Abstract

Algorithms from the book Nash (1979) are implemented in a variety of programming languages including Fortran, BASIC, Pascal, Python and R.

Overview of this document

A companion document **Overview of Nashlib and its Implementations** describes the process and computing environments for the implementation of Nashlib algorithms. This document gives comments and/or details relating to implementations of the algorithms themselves.

Note that some discussion of the reasoning behind certain choices in algorithms or implementations are given in the Overview document.

Algorithms 1 and 2 – one-sided SVD and least squares solution

These were two of the first algorithms to interest the first author in compact codes. At the time (1973-1978) he was working at Agriculture Canada in support of econometric modeling. More or less "regular" computers required accounts linked to official projects, but there was a time-shared Data General NOVA that offered 4K to 7K byte working spaces for data and programs in interpreted BASIC. BASIC of a very similar dialect was available also on an HP 9830 calculator. On these machines, availability of a terminal or the calculator was the only limitation to experimentation with recent innovations in algorithms. In particular, a lot of modeling was done with linear least squares regression, mostly using the traditional normal equations. The singular value decomposition and other methods such as the Householder, Givens or Gram-Schmidt approaches to the QR matrix decomposition were relatively recent innovations. However, the code for the Golub-Kahan SVD was rather long for both the hardware and the BASIC language. Instead, a one-sided Jacobi method was developed from ideas of Hestenes (1958) and Chartres (1962). Some work by Kaiser (1972) was also observed. Later workers have generally credited Hestenes with this approach, and he certainly wrote about it, but we (JN) suspect strongly that he never actually attempted an implementation. In a conversation at a conference, Chartres said that some experiments were tried, but that he believed no production usage occurred. We must remember that access to computers until the 1970s was quite difficult.

The method published in Nash (1975) and later revised in Nash and Shlien (1987) ignored some advice that Jacobi rotations should not use angles greater than $\pi/4$ (see Forsythe and Henrici (1960)). This allowed of a cyclic process that not only developed a form of the decomposition, but also sorted it to effectively present the singular values in descending order of size. This avoided extra program code of about half the length of the svd routine.

About 2 decades after Nash (1975), there was renewed interest in one-sided Jacobi methods, but rather little acknowledgment of the earlier development, and much more complicated codes. ?? How far to reference more recent developments??

Fortran

Listing

Note that this is a single precision code. Very few modern calculations are carried out at this precision. Moreover, the dialect of Fortran (Fortran 77) is now decidedly old-fashioned, though it compiles and executes just fine.

```
C&&& A1-2
C  TEST ALGS. 1 & 2   J.C. NASH   JULY 1978, APRIL 1989
C  USES FRANK MATRIX COLUMNS
      LOGICAL ESVD,NTOL
      INTEGER N,ND,IPOS(10),NVAR,MD,I,J,K,YPOS,M
      REAL A(30,10),D(30,11),G(30),X(10),Z(10),Y(30),Q,V(10,10),EPS
      EXTERNAL FRANKM
C  I/O CHANNELS
      NIN=5
      NOUT=6
      ND=10
      MD=30
      D(1,1)=5
      D(1,2)=1.0E-6
      D(1,3)=1
      Y(1)=1
      D(2,1)=6
      D(2,2)=0.999999
      D(2,3)=1
      Y(2)=2
```

```
      D(3,1)=7
      D(3,2)=2.00001
      D(3,3)=1
      Y(3)=3
      D(4,1)=8
      D(4,2)=2.9999
      D(4,3)=1
      Y(4)=4
      N=3
      M=4
      DO 30 J=1,N
      DO 25 I=1,M
        A(I,J)=D(I,J)
 25   CONTINUE
 30   CONTINUE
      ESVD=.FALSE.
      WRITE(NOUT,955)(Y(I),I=1,M)
 955  FORMAT(1H ,5E16.8)
      NTOL=.FALSE.
      Q = 1e-5
      WRITE(NOUT,956)Q
 956  FORMAT(' SING. VALS. .LE.',E16.8,'  ARE PRESUMED ZERO')
      IF(Q.LT.0.0) STOP
C  "MACHINE PRECISION" VALUE
      EPS=1E-6
      CALL A2LSVD(M,N,A,MD,EPS,V,ND,Z,NOUT,Y,G,X,Q,ESVD,NTOL)
      WRITE(NOUT,957)(J,X(J),J=1,N)
 957  FORMAT(' X(',I3,')=',1PE16.8)
      STOP
      END
      SUBROUTINE OUT(A,NA,N,NP,NOUT)
C  J.C. NASH   JULY 1978, APRIL 1989
      INTEGER NA,N,NOUT,I,J
      REAL A(NA,NP)
      DO 20 I=1,N
        WRITE(NOUT,951)I
 951    FORMAT(' ROW',I3)
        WRITE(NOUT,952)(A(I,J),J=1,NP)
 952    FORMAT(1H ,1P5E16.8)
 20   CONTINUE
      RETURN
      END
      SUBROUTINE FRANKM(M,N,A,NA)
C  J.C. NASH   JULY 1978, APRIL 1989
      INTEGER M,N,NA,I,J
C  INPUTS FRANK MATRIX M BY N INTO A
      REAL A(NA,N)
      DO 20 I=1,M
        DO 10 J=1,N
          A(I,J)=AMIN0(I,J)
 10     CONTINUE
 20   CONTINUE
      RETURN
```

```
      END
      SUBROUTINE A3PREP(M,N1,A,NA,AIN)
C  PREPARE A3 TEST
C  J.C. NASH   JULY 1978, APRIL 1989
C  MATRIX M BY N=N1-1 IS INPUT VIA SUBROUTINE AIN
C  COL. N1 IS SET TO SUM OF OTHER COLS.  - UNIT SOLUTION ELEMENTS
C  BUT ONLY IF M=N - OTHERWISE SIMPLY INPUT MATRIX
C  NA = FIRST DIMENSION OF A
      INTEGER M,N1,NA,N,J,I
      REAL A(NA,N1),S
      N=N1-1
      CALL AIN(M,N,A,NA)
      IF(M.NE.N)RETURN
      DO 40 I=1,N
        S=0.0
        DO 30 J=1,N
          S=S+A(I,J)
  30    CONTINUE
        A(I,N1)=S
  40  CONTINUE
      RETURN
      END
      SUBROUTINE A1SVD(M,N,A,NA,EPS,V,NV,Z,IPR)
C  ALGORITHM 1  SINGULAR VALUE DECOMPOSITION BY COLUMN ORTHOGONA-
C     LISATION VIA PLANE ROTATIONS
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  M BY N  MATRIX A  IS DECOMPOSED TO  U*Z*VT
C   A    =   ARRAY CONTAINING A (INPUT),  U (OUTPUT)
C   NA   =   FIRST DIMENSION OF A
C   EPS  =   MACHINE PRECISION
C   V    =   ARRAY IN WHICH ORTHOGAONAL MATRIX V IS ACCUMULATED
C   NV   =   FIRST DIMENSION OF V
C   Z    =   VECTOR OF SINGULAR VALUES
C  IPR   =  PRINT CHANNEL   IF IPR.GT.0  THEN PRINTING
C  STEP 0
      INTEGER M,N,J1,N1,COUNT
      REAL A(NA,N),V(NV,N),Z(N),EPS,TOL,P,Q,R,VV,C,S
C  UNDERFLOW AVOIDANCE STRATEGY
      REAL SMALL
      SMALL=1.0E-36
C  ABOVE IS VALUE FOR IBM
      TOL=N*N*EPS*EPS
      DO 6 I=1,N
        DO 4 J=1,N
          V(I,J)=0.0
   4    CONTINUE
      V(I,I)=1.0
   6  CONTINUE
      N1=N-1
C  STEP 1
  10  COUNT=N*(N-1)/2
C  STEP 2
      DO 140 J=1,N1
```

```
C  STEP 3
        J1=J+1
        DO 130 K=J1,N
C  STEP 4
          P=0.0
          Q=0.0
          R=0.0
C  STEP 5
          DO 55 I=1,M
            IF(ABS(A(I,J)).GT.SMALL.AND.ABS(A(I,K)).GT.SMALL)
     #         P=P+A(I,J)*A(I,K)
            IF(ABS(A(I,J)).GT.SMALL)Q=Q+A(I,J)**2
            IF(ABS(A(I,K)).GT.SMALL)R=R+A(I,K)**2
C           P=P+A(I,J)*A(I,K)
C           Q=Q+A(I,J)**2
C           R=R+A(I,K)**2
 55       CONTINUE
C  STEP 6
          IF(Q.GE.R)GOTO 70
          C=0.0
          S=1.0
          GOTO 90
C  STEP 7
 70       IF(R.LE.TOL)GOTO 120
          IF((P*P)/(Q*R).LT.TOL)GOTO 120
C  STEP 8
          Q=Q-R
          VV=SQRT(4.0*P**2+Q**2)
          C=SQRT((VV+Q)/(2.0*VV))
          S=P/(VV*C)
C  STEP 9
 90       DO 95 I=1,M
            R=A(I,J)
            A(I,J)=R*C+A(I,K)*S
            A(I,K)=-R*S+A(I,K)*C
 95       CONTINUE
C  STEP 10
          DO 105 I=1,N
            R=V(I,J)
            V(I,J)=R*C+V(I,K)*S
            V(I,K)=-R*S+V(I,K)*C
105       CONTINUE
C  STEP 11
          GOTO 130
120       COUNT=COUNT-1
C  STEP 13
130     CONTINUE
C  STEP 14
140   CONTINUE
C  STEP 15
      IF(IPR.GT.0)WRITE(IPR,964)COUNT
964   FORMAT(1H ,I4,10H ROTATIONS)
      IF(COUNT.GT.0)GOTO 10
```

```
C  STEP 16
      DO 220 J=1,N
C  STEP 17
        Q=0.0
C  STEP 18
        DO 185 I=1,M
            Q=Q+A(I,J)**2
 185    CONTINUE
C  STEP 19
        Q=SQRT(Q)
        Z(J)=Q
        IF(IPR.GT.0)WRITE(IPR,965)J,Q
 965    FORMAT( 4H SV(,I3,2H)=,1PE16.8)
C  STEP 20
        IF(Q.LT.TOL)GOTO 220
C  STEP 21
        DO 215 I=1,M
          A(I,J)=A(I,J)/Q
 215    CONTINUE
C  STEP 22
 220  CONTINUE
      RETURN
      END
      SUBROUTINE A2LSVD(M,N,A,NA,EPS,V,NV,Z,IPR,Y,G,X,Q,ESVD,NTOL)
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  SAME COMMENTS AS SUBN A1SVD EXCEPT FOR
C   G    =  WORKING VECTOR IN N ELEMENTS
C   Y    =  VECTOR CONTAINING M VALUES OF DEPENDENT VARIABLE
C   X    =  SOLUTION VECTOR
C  Q    =  TOLERANCE FOR SINGULAR VALUES. THOSE .LE. Q TAKEN AS ZERO.
C ESVD  =  LOGICAL FLAG SET .TRUE. IF SVD ALREADY EXISTS IN A,Z,V
C NTOL  =  LOGICAL FLAG SET .TRUE. IF ONLY NEW TOLERANCE Q.
      LOGICAL ESVD,NTOL
      INTEGER M,N,IPR,I,J
      REAL A(NA,N),V(NV,N),Z(N),Y(M),G(N),X(N),S,Q
C  STEP 1
      IF(NTOL)GOTO 41
      IF(.NOT.ESVD)CALL A1SVD(M,N,A,NA,EPS,V,NV,Z,IPR)
      IF(IPR.GT.0)WRITE(IPR,965)(J,Z(J),J=1,N)
  965  FORMAT(16H SINGULAR VALUE(,I3,2H)=,1PE16.8)
C  STEP 2 VIA SUBROUTINE CALL
C  ALTERNATIVE WITHOUT G
C  NO STEP 3
C  STEP 3  UT*Y=G
      DO 36 J=1,N
        S=0.0
        DO 34 I=1,M
          S=S+A(I,J)*Y(I)
  34    CONTINUE
        G(J)=S
  36  CONTINUE
C  STEP 4
  41  IF(Q.LT.0.0)STOP
```

```
C  STEP 5
      DO 56 J=1,N
        S=0.0
        DO 54 I=1,N
          IF(Z(I).GT.Q)S=S+V(J,I)*G(I)/Z(I)
   54   CONTINUE
        X(J)=S
  56  CONTINUE
C  STEP 6
C  NEW TOLERANCE VIA NEW CALL
      RETURN
      END
```

Example output

```
gfortran ../fortran/dr0102.f
mv ./a.out ../fortran/dr0102.run
../fortran/dr0102.run < ../fortran/dr0102.in
```

```
##    0.10000000E+01  0.20000000E+01  0.30000000E+01  0.40000000E+01
##  SING. VALS. .LE.  0.99999997E-05  ARE PRESUMED ZERO
##     3 ROTATIONS
##     3 ROTATIONS
##     1 ROTATIONS
##     0 ROTATIONS
##  SV(  1)=  1.37529879E+01
##  SV(  2)=  1.68960798E+00
##  SV(  3)=  1.18504076E-05
##  SINGULAR VALUE(  1)=  1.37529879E+01
##  SINGULAR VALUE(  2)=  1.68960798E+00
##  SINGULAR VALUE(  3)=  1.18504076E-05
##  X(  1)=  1.00434840E+00
##  X(  2)= -4.34857607E-03
##  X(  3)= -4.02174187E+00
```

Special implementations

Most singular value decomposition codes are much, much more complicated than Algorithm 1 of the Nashlib collection. For some work on the magnetic field of Jupiter for NASA, Sidey Timmins has used an extended (quad) precision version of the method. One of us (JN) has converted an updated algorithm (Nash and Shlien (1987)) to the Fortran 95 dialect so the multiple precision FM Fortran tools of David M. Smith (see http://dmsmith.lmu.build/).

?? include this code and example in the repo??

BASIC

Listing

```
5 PRINT "dr0102.bas -- Nashlib Alg 01 and 02 driver"
10 PRINT "from ENHSVA APR 7 80 -- MOD 850519, remod 210113"
20 LET E1=1.0E-7
30 PRINT "ONE SIDED TRANSFORMATION METHOD FOR REGRESSIONS VIA"
40 PRINT "THE SINGULAR VALUE DECOMPOSITION -- J.C.NASH 1973,79"
150 LET M=4
```

```
160 LET N=3
210 DIM Y(M,N+1),A(M,N),T(N,N),G(N),X(N),Z(N),U(N),B(M)
220 DIM F$(10)
230 LET F$="K"
236 PRINT "Prep matrix and RHS"
240 LET Y(1,1)=5
241 LET Y(1,2)=1.0E-6
242 LET Y(1,3)=1
243 LET B(1)=1
250 LET Y(2,1)=6
251 LET Y(2,2)=0.999999
252 LET Y(2,3)=1
253 LET B(2)=2
260 LET Y(3,1)=7
261 LET Y(3,2)=2.00001
262 LET Y(3,3)=1
263 LET B(3)=3
270 LET Y(4,1)=8
271 LET Y(4,2)=2.9999
272 LET Y(4,3)=1
273 LET B(4)=4
500 FOR I=1 TO M
510 FOR J=1 TO N-1
520 LET A(I,J)=Y(I,J)
530 NEXT J
535 quit
540 LET A(I,N)=E3
550 NEXT I
560 LET E2=N*N*E1*E1
570 PRINT
580 FOR I=1 TO N
590 FOR J=1 TO N
600 LET T(I,J)=0
610 NEXT J
620 LET T(I,I)=1
630 NEXT I
640 LET I9=0
650 IF N=1 THEN GOTO 1150
660 LET N2=N*(N-1)/2
670 LET N1=N-1
680 LET N9=N2
690 LET I9=I9+1
700 FOR J=1 TO N1
710 LET J1=J+1
720 FOR K=J1 TO N
730 LET P=0
740 LET Q=0
750 LET R=0
760 FOR I=1 TO M
770 LET P=P+A(I,J)*A(I,K)
780 LET Q=Q+A(I,J)*A(I,J)
790 LET R=R+A(I,K)*A(I,K)
800 NEXT I
```

```
810 IF Q>=R THEN GOTO 850
820 LET C=0
830 LET S=1
840 GOTO 920
850 IF (Q*R)<=0 THEN GOTO 1040
860 IF P*P/(Q*R)<E2 THEN GOTO 1040
870 LET Q=Q-R
880 LET P=2*P
890 LET V1=SQR(P*P+Q*Q)
900 LET C=SQR((V1+Q)/(2*V1))
910 LET S=P/(2*V1*C)
920 FOR I=1 TO M
930 LET V1=A(I,J)
940 LET A(I,J)=V1*C+A(I,K)*S
950 LET A(I,K)=-V1*S+A(I,K)*C
960 NEXT I
970 FOR I=1 TO N
980 LET V1=T(I,J)
990 LET T(I,J)=V1*C+T(I,K)*S
1000 LET T(I,K)=-V1*S+T(I,K)*C
1010 NEXT I
1020 LET N9=N2
1030 GOTO 1060
1040 LET N9=N9-1
1050 IF N9=0 THEN GOTO 1150
1051 REM ?? GOTO was EXIT for NS BASIC
1060 NEXT K
1070 NEXT J
1080 PRINT "SWEEP",I9,
1090 IF O1>0 THEN PRINT #O1,"SWEEP ",I9," ",
1100 IF 6*INT(I9/6)<>I9 THEN GOTO 680
1110 IF O1>0 THEN PRINT #O1
1120 IF I9>=30 THEN GOTO 1150
1130 PRINT
1140 GOTO 680
1150 PRINT
1160 IF O1>0 THEN PRINT #O1
1170 PRINT "CONVERGENCE AT SWEEP ",I9
1180 IF O1>0 THEN PRINT #O1,"CONVERGENCE AT SWEEP ",I9
1190 FOR J=1 TO N
1200 LET Q=0
1210 FOR I=1 TO M
1220 LET Q=Q+A(I,J)^2
1230 NEXT I
1240 LET Q=SQR(Q)
1250 IF Q=0 THEN GOTO 1290
1260 FOR I=1 TO M
1270 LET A(I,J)=A(I,J)/Q
1280 NEXT I
1290 LET Z(J)=Q
1300 NEXT J
1310 PRINT
1320 PRINT "SINGULAR VALUES"
```

```
1340 FOR J=1 TO N
1350 PRINT Z(J),
1370 IF 5*INT(J/5)<>J THEN GOTO 1400
1380 PRINT
1400 NEXT J
1410 PRINT
1430 PRINT "VARIABLE # OF REGRESSAND",
1440 INPUT M2
1450 IF M2<=0 THEN GOTO 350
1470 LET S1=0
1480 FOR I=1 TO M
1490 LET S1=S1+(Y(I,M2)-E3*Y(M+1,M2))^2
1500 NEXT I
1510 FOR J=1 TO N
1520 LET S=0
1530 FOR I=1 TO M
1540 LET S=S+A(I,J)*Y(I,M2)
1550 NEXT I
1560 LET G(J)=S
1570 NEXT J
1580 PRINT "ENTER TOLERANCE FOR ZERO",
1590 INPUT Q
1600 IF Q<0 THEN GOTO 1410
1610 PRINT "SINGULAR VALUES <=",Q," ARE TAKEN AS 0"
1630 LET R=0
1640 FOR I=1 TO N
1650 LET V1=0
1660 LET S=0
1670 LET P=0
1680 FOR K=1 TO N
1690 LET C=0
1700 IF Z(K)<=Q THEN GOTO 1730
1710 LET C=1/Z(K)
1720 LET V1=V1+1
1730 LET S=S+C*T(I,K)*G(K)
1740 LET P=P+(C*T(I,K))^2
1750 NEXT K
1760 LET U(I)=P
1770 LET X(I)=S
1780 LET R=R+S*S
1790 NEXT I
1800 LET X(N)=X(N)*E3
1810 PRINT
1820 PRINT "RESIDUALS"
1840 LET C=0
1850 LET S2=0
1860 FOR I=1 TO M
1870 LET S=Y(I,M2)-X(N)
1880 FOR K=1 TO N-1
1890 LET S=S-Y(I,W(K))*X(K)
1900 NEXT K
1910 PRINT S,
1930 IF 5*INT(I/5)<>I THEN GOTO 1960
```

```
1940 PRINT
1960 LET C=C+S*S
1970 IF I=1 THEN GOTO 1990
1980 LET S2=S2+(S-S3)^2
1990 LET S3=S
2000 NEXT I
2010 PRINT
2020 LET P=0
2040 IF M<=V1 THEN GOTO 2060
2050 LET P=C/(M-V1)
2060 PRINT M-V1," DEGREES OF FREEDOM"
2080 REM PRINT
2090 PRINT "SOLUTION VECTOR - CONSTANT LAST"
2110 FOR I=1 TO N
2120 LET V1=SQR(P*U(I))
2130 PRINT "X(",W(I),")=",X(I),"  STD.ERR.=",V1,
2140 IF O1>0 THEN PRINT #O1,"X(",W(I),")=",X(I),"  STD.ERR.=",V1,
2150 IF V1<=0 THEN GOTO 2180
2160 PRINT "  T=",ABS(X(I)/V1),
2170 IF O1>0 THEN PRINT #O1,"  T=",ABS(X(I)/V1),
2180 PRINT
2190 IF O1>0 THEN PRINT #O1
2200 NEXT I
2210 PRINT "SUM OF SQUARES",C," SIGMA^2",P
2220 IF O1>0 THEN PRINT #O1,"SUM OF SQUARES",C," SIGMA^2",P
2230 PRINT "NORM OF SOLUTION",SQRT(R)
2240 IF O1>0 THEN PRINT #O1,"NORM OF SOLUTION",SQRT(R)
2250 PRINT "R SQUARED=",1-C/S1," DURBIN-WATSON STAT.=",S2/C
2260 IF O1>0 THEN PRINT #O1,"R SQUARED=",1-C/S1," DURBIN-WATSON STAT.=",S2/C
2270 PRINT
2280 IF O1>0 THEN PRINT #O1
2290 GOTO 1580
2300 REM GET SERIES FROM FILE
2310 PRINT "FILENAME OR 'KEYBOARD' OR 'K'",
2320 INPUT G$
2330 IF LEN(G$)>0 THEN LET F$=G$
2331 REM DEFAULTS TO LAST SETTING
2340 PRINT "DATA FROM FILE :",F$
2350 IF F$="KEYBOARD" THEN 2420
2360 IF F$<>"K" THEN 2460
2370 PRINT
2380 PRINT "ENTER SERIES"
2390 FOR I=1 TO M
2400 INPUT1 Y(I,J)
2410 IF 5*INT(I/5)=I THEN PRINT
2420 NEXT I
2430 PRINT
2440 IF O1>0 THEN GOSUB 2860
2450 RETURN
2460 IF FILE(F$)=3 THEN 2490
2470 PRINT "FILE NOT FOUND OR OF WRONG TYPE"
2480 GOTO 2310
2490 OPEN #1,F$
```

```
2500 PRINT "SERIES NAME OR #",
2510 INPUT X$
2520 IF X$(1,1)="#" THEN 2770
2530 IF TYP(1)=0 THEN 2740
2540 IF TYP(1)=1 THEN 2570
2550 READ #1,C
2560 GOTO 2530
2570 READ #1,Y$
2580 IF X$<>Y$ THEN 2530
2590 I=0
2600 PRINT "SERIES:",Y$
2610 IF O1>0 THEN PRINT #O1,"SERIES:",Y$
2620 IF TYP(1)<>2 THEN 2690
2630 IF I=M THEN 2690
2640 I=I+1
2650 READ#1,Y(I,J)
2660 PRINT Y(I,J),
2670 IF 5*INT(I/5)=I THEN PRINT
2680 GOTO 2620
2690 PRINT
2700 PRINT "END OF SERIES  ",I," DATA POINTS"
2710 IF O1>0 THEN GOSUB 2860
2720 CLOSE #1
2730 RETURN
2740 PRINT "END OF FILE"
2750 CLOSE #1
2760 GOTO 2310
2770 X$=X$(2)
2780 P1=VAL(X$)
2790 J=0
2800 IF TYP(1)=0 THEN 2740
2810 IF TYP(1)=1 THEN 2840
2820 READ#1,C
2830 GOTO 2800
2840 J=J+1
2850 READ#1,Y$
2860 FOR I=1 TO M
2870 PRINT #O1,Y(I,J),
2880 IF 5*INT(I/5)=I THEN PRINT #O1
2890 NEXT I
2900 PRINT #O1
2910 RETURN
```

Example output

```
bwbasic ../BASIC/dr0102.bas
echo "done"
```

```
## Bywater BASIC Interpreter/Shell, version 2.20 patch level 2
## Copyright (c) 1993, Ted A. Campbell
## Copyright (c) 1995-1997, Jon B. Volkoff
##
## dr0102.bas -- Nashlib Alg 01 and 02 driver
## from ENHSVA APR 7 80 -- MOD 850519, remod 210113
```

```
## ONE SIDED TRANSFORMATION METHOD FOR REGRESSIONS VIA
## THE SINGULAR VALUE DECOMPOSITION -- J.C.NASH 1973,79
## Prep matrix and RHS
##
## done
```

Pascal

Listing

```
Program runsvd(input,output);
{dr0102.pas == Calculation of Singular values and vectors of an arbitrary
          real matrix, solution of linear least squares approximation
          problem.

  Modifies a method due to Kaiser.  See Nash and Shlien (1987): Simple
  algorithms for the partial singular value decomposition.  Computer
  Journal, vol.30, pp.268-275.

          Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}


{constype.def ==
  This file contains various definitions and type statements which are
  used throughout the collection of "Compact Numerical Methods".  In many
  cases not all definitions are needed, and users with very tight memory
  constraints may wish to remove some of the lines of this file when
  compiling certain programs.

  Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}
uses Dos, Crt; {Turbo Pascal 5.0 Modules}
{ 1. Interrupt, Unit, Interface, Implementation, Uses are reserved words now.}
{ 2. System,Dos,Crt are standard unit names in Turbo 5.0.}

const
  big = 1.0E+35;    {a very large number}
  Maxconst = 25;    {Maximum number of constants in data record}
  Maxobs = 100;     {Maximum number of observations in data record}
  Maxparm  = 25;    {Maximum number of parameters to adjust}
  Maxvars = 10;     {Maximum number of variables in data record}
  acctol = 0.0001;  {acceptable point tolerance for minimisation codes}
  maxm = 20;        {Maximum number or rows in a matrix}
  maxn = 20;        {Maximum number of columns in a matrix}
  maxmn = 40;       {maxn+maxm, the number of rows in a working array}
  maxsym = 210;     {maximum number of elements of a symmetric matrix
              which need to be stored = maxm * (maxm + 1)/2 }
  reltest = 10.0;   {a relative size used to check equality of numbers.
              Numbers x and y are considered equal if the
```

```
              floating-point representation of reltest+x equals
              that of reltest+y.}
  stepredn = 0.2;   {factor to reduce stepsize in line search}
  yearwrit = 1990;  {year in which file was written}

type
  str2 = string[2];
  rmatrix = array[1..maxm, 1..maxn] of real; {a real matrix}
  wmatrix = array[1..maxmn, 1..maxn] of real; {a working array, formed
                  as one real matrix stacked on another}
  smatvec = array[1..maxsym] of real; {a vector to store a symmetric matrix
              as the row-wise expansion of its lower triangle}
  rvector = array[1..maxm] of real;  {a real vector. We will use vectors
              of m elements always. While this is NOT space efficient,
              it simplifies program codes.}
  cgmethodtype= (Fletcher_Reeves,Polak_Ribiere,Beale_Sorenson);
    {three possible forms of the conjugate gradients updating formulae}
  probdata = record
          m      : integer; {number of observations}
          nvar   : integer; {number of variables}
          nconst: integer; {number of constants}
          vconst: array[1..Maxconst] of real;
          Ydata : array[1..Maxobs, 1..Maxvars] of real;
          nlls  : boolean; {true if problem is nonlinear least squares}
        end;
{
  NOTE: Pascal does not let us define the work-space for the function
  within the user-defined code.  This is a weakness of Pascal for this
  type of work.
}
var {global definitions}
  banner     : string[80]; {program name and description}

function calceps:real;
{calceps.pas ==
  This function returns the machine EPSILON or floating point tolerance,
  the smallest positive real number such that 1.0 + EPSILON > 1.0.
  EPSILON is needed to set various tolerances for different algorithms.
  While it could be entered as a constant, I prefer to calculate it, since
  users tend to move software between machines without paying attention to
  the computing environment. Note that more complete routines exist.
}
var
  e,e0: real;
  i: integer;
begin {calculate machine epsilon}
  e0 := 1; i:=0;
  repeat
    e0 := e0/2; e := 1+e0;  i := i+1;
  until (e=1.0) or (i=50); {note safety check}
  e0 := e0*2;
{ Writeln('Machine EPSILON =',e0);}
  calceps:=e0;
```

```
end; {calceps}

function resids(nRow, nCol: integer; A : rmatrix;
          Y: rvector; Bvec : rvector; print : boolean):real;
{resids.pas
  == Computes residuals and , if print is TRUE, displays them 7
    per line for the linear least squares problem. The sum of
    squared residuals is returned.

    residual vector = A * Bvec - Y
}
var
i, j: integer;
t1, ss : real;

begin
  if print then
  begin
    writeln('Residuals');
  end;
  ss:=0.0;
  for i:=1 to nRow do
  begin
    t1:=-Y[i]; {note form of residual is residual = A * B - Y }
    for j:=1 to nCol do
      t1:=t1+A[i,j]*Bvec[j];
    ss:=ss+t1*t1;
    if print then
    begin
      write(t1:10,' ');
      if (i = 7 * (i div 7)) and (i<nRow) then writeln;
    end;
  end; {loop on i}
  if print then
  begin
    writeln;
    writeln('Sum of squared residuals =',ss);
  end;
  resids:=ss
end; {resids.pas == residual calculation for linear least squares}


Procedure matcopy(nRow ,nCol: integer; A: rmatrix; var B:wmatrix);
{matcopy.pas
  -- copies matrix A, nRow by nCol, into matrix B }
var i,j: integer;
begin
  for i:=1 to nRow do
    for j:=1 to nCol do
      B[i,j]:=A[i,j];
end;{matcopy.pas}
```

```
Procedure PrtSVDResults( nRow, nCol:integer;
                U, V: rmatrix; Z: rvector);
{psvdres.pas
  == routine to display svd results and print them to confile
}
var
  i, j : integer;

begin
  writeln(' Singular values and vectors:');
  for j := 1 TO nCol do
  begin
    writeln('Singular value (',j,') =', Z[j]);
    writeln('Principal coordinate (U):');
    for i := 1 to nRow do
    begin
      write(U[i,j]:10:7);
      if (7 * (i div 7) = i) and (i<nRow) then writeln;
    end;
    writeln;
    writeln('Principal component (V):');
    for i:=1 to nCol do
    begin
      write(V[i,j]:10:7);
      if (7 * (i div 7) = i) and (i<nCol) then writeln;
    end;
    writeln;
  end;
end; {psvdres == print svd results via procedure PrtSVDResults }


Procedure svdtst( A, U, V: rmatrix; Z: rvector;
              nRow, UCol, VCol: integer);
{svdtst.pas
  == This routine tests the results of a singular value
  decomposion calculation.  The matrix A is presumed to contain
  the matrix of which the purported decomposition is

        U  Z  V-transpose

  This routine tests column orthogonality of U and V,
  row orthogonality of V, and the reconstruction suggested
  by the decomposition. It does not carry out the tests of
  the Moore-Penrose inverse A+, which can be computed as

      A+ :=  V  Z  U-transpose.

  FORTRAN codes for the conditions

          A+ A A+  = ? =  A+
          A  A+ A  = ? =  A
          (A+  A)-transpose  = ? =  A+ A
          (A  A+)-transpose  = ? =  A  A+
```

```
  are given in Nash, J.C. and Wang, R.L.C. (1986)
}

var
  i,j,k:integer;
  t1: real;
  imax, jmax: integer;
  valmax: real;

begin
  writeln('Column orthogonality of U');
  valmax:=0.0;
  imax:=0;
  jmax:=0;
  for i:=1 to UCol do
  begin
    for j:=i to UCol do
    begin
      t1:=0.0; {accumulate inner products}
      if i=j then t1:=-1;
      for k:=1 to nRow do t1:=t1+U[k,i]*U[k,j];
      if abs(t1)>abs(valmax) then
      begin
        imax:=i; jmax:=j; valmax:=t1;
      end;
    end;
  end;
  writeln('Largest inner product is ',imax,',',jmax,'=',valmax);
  writeln('Row orthogonality of U (NOT guaranteed in svd)');
  valmax:=0.0;
  imax:=0;
  jmax:=0;
  for i:=1 to nRow do
  begin
    for j:=i to nRow do
    begin
      t1:=0.0; {accumulate inner products}
      if i=j then t1:=-1;
      for k:=1 to UCol do t1:=t1+U[i,k]*U[j,k];
      if abs(t1)>abs(valmax) then
      begin
        imax:=i; jmax:=j; valmax:=t1;
      end;
    end;
  end;
  writeln('Largest inner product is ',imax,',',jmax,'=',valmax);
  writeln('Column orthogonality of V');
  valmax:=0.0;
  imax:=0;
  jmax:=0;
  for i:=1 to VCol do
  begin
    for j:=i to VCol do
```

```
    begin
      t1:=0.0; {accumulate inner products}
      if i=j then t1:=-1.0;
      for k:=1 to VCol do t1:=t1+V[k,i]*V[k,j];
      if abs(t1)>abs(valmax) then
      begin
        imax:=i; jmax:=j; valmax:=t1;
      end;
    end;
  end;
  writeln('Largest inner product is ',imax,',',jmax,'=',valmax);
  writeln('Row orthogonality of V');
  valmax:=0.0;
  imax:=0;
  jmax:=0;
  for i:=1 to VCol do
  begin
    for j:=i to VCol do
    begin
      t1:=0.0; {accumulate inner products}
      if i=j then t1:=-1;
      for k:=1 to VCol do t1:=t1+V[i,k]*V[j,k];
      if abs(t1)>abs(valmax) then
      begin
        imax:=i; jmax:=j; valmax:=t1;
      end;
    end;
  end;
  writeln('Largest inner product is ',imax,',',jmax,'=',valmax);
  writeln('Reconstruction of initial matrix');
  valmax:=0.0;
  imax:=0;
  jmax:=0;
  for i:=1 to nRow do
  begin
    for j:=1 to VCol do
    begin
      t1:=0;
      for k:=1 to VCol do
        t1:=t1+U[i,k]*Z[k]*V[j,k]; { U * S * V-transpose}
      {writeln('A[',i,',',j,']=',A[i,j],' Recon. =',t1,'  error=',A[i,j]-t1);}
      if abs(A[i,j]-t1)>abs(valmax) then
      begin
        imax:=i; jmax:=j; valmax:=A[i,j]-t1;
      end;
    end;
  end;
  writeln('Largest error is ',imax,',',jmax,'=',valmax);
end; {svdtst.pas}


{I matrixin.pas} {input or generate a matrix of reals}
{I vectorin.pas} {input or generate a vector of reals}
```

```
procedure NashSVD(nRow, nCol: integer;
               var W: wmatrix;
               var Z: rvector);

var
  i, j, k, EstColRank, RotCount, SweepCount, slimit : integer;
  eps, e2, tol, vt, p, x0, y0, q, r, c0, s0, d1, d2 : real;

procedure rotate;
var
  ii : integer;

begin
  for ii := 1 to nRow+nCol do
  begin
    D1 := W[ii,j]; D2 := W[ii,k];
    W[ii,j] := D1*c0+D2*s0; W[ii,k] := -D1*s0+D2*c0
  end;
end;

begin
  writeln('alg01.pas -- NashSVD');
  eps := Calceps;
  slimit := nCol div 4;  if slimit<6 then slimit := 6;

  SweepCount := 0;
  e2 := 10.0*nRow*eps*eps;
  tol := eps*0.1;

  EstColRank := nCol; ;

  for i := 1 to nCol do
    begin
    for j := 1 to nCol do
      W[nRow+i,j] := 0.0;
    W[nRow+i,i] := 1.0;
  end;

  repeat
    RotCount := EstColRank*(EstColRank-1) div 2;
    SweepCount := SweepCount+1;

    for j := 1 to EstColRank-1 do
    begin
      for k := j+1 to EstColRank do
      begin
        p := 0.0;  q := 0.0; r := 0.0;
        for i := 1 to nRow do
        begin
          x0 := W[i,j]; y0 := W[i,k];
          p := p+x0*y0; q := q+x0*x0;  r := r+y0*y0;
        end;
```

```
        Z[j] := q; Z[k] := r;

        if q >= r then
        begin
          if (q<=e2*Z[1]) or (abs(p)<= tol*q) then RotCount := RotCount-1

          else
          begin
            p := p/q; r := 1-r/q; vt := sqrt(4*p*p + r*r);
            c0 := sqrt(0.5*(1+r/vt)); s0 := p/(vt*c0);
            rotate;
          end
        end
        else
        begin

          p := p/r; q := q/r-1; vt := sqrt(4*p*p + q*q);
          s0 := sqrt(0.5*(1-q/vt));
          if p<0 then s0 := -s0;
          c0 := p/(vt*s0);
          rotate;
        end;

      end;
    end;
    writeln('End of Sweep #', SweepCount,
            '-  no. of rotations performed =', RotCount);
    while (EstColRank >= 3) and (Z[EstColRank] <= Z[1]*tol + tol*tol)
          do EstColRank := EstColRank-1;
  until (RotCount=0) or (SweepCount>slimit);
  if (SweepCount > slimit) then writeln('**** SWEEP LIMIT EXCEEDED');
end;


procedure svdlss(nRow, nCol: integer;
                 W : wmatrix;
                 Y: rvector;
                 Z : rvector;
                 A : rmatrix;
                 var Bvec: rvector;
                 q : real);

var
 i, j, k : integer;
 s : real;

begin
  writeln('alg02.pas == svdlss');
{  write('Y:');
  for i := 1 to nRow do
  begin
    write(Y[i],' ');
  end;
```

```
  writeln;

  for i := 1 to (nRow+nCol) do
  begin
     write('W row ',i,':');
     for j:= 1 to nCol do
     begin
       write(W[i,j],' ');
     end;
     writeln;
   end;
}
{    writeln('Singular values');
    for j := 1 to nCol do
    begin
      write(Z[j]:18,' ');
      if j=4 * (j div 4) then writeln;
    end;
    writeln;
}
    if q>=0.0 then
    begin
    q := q*q;
      for i := 1 to nCol do
      begin
        s := 0.0;
        for j := 1 to nCol do
        begin
          for k := 1 to nRow do
          begin
            if Z[j]>q then
              s := s + W[i+nRow,j]*W[k,j]*Y[k]/Z[j];
                       { V   S+   U'  y }

          end;
        end;
        Bvec[i] := s;
      end;
      writeln('Least squares solution');
      for j := 1 to nCol do
      begin
        write(Bvec[j]:12,' ');
        if j=5 * (j div 5) then writeln;
      end;
      writeln;
      s := resids(nRow, nCol, A, Y, Bvec, true);
    end;
end;

{main program}
var
  nRow, nCol : integer;
  A, V, U : rmatrix;
```

```
  W : wmatrix; {a working matrix which will contain U Zd in the
    upper nRow rows, and V in the bottom nCol rows, where Zd
    is the diagonal matrix of singular values.  That is, W
    becomes

                 (  U  Zd )
                 (        )
                 (    V   )

  }
  Z, Zsq : rvector; {Z will contain either the squares of singular
          values or the singular values themselves}
  Y : rvector; {Y will contain the 'right hand side' of the
          least squares problem, i.e.  the vector to be
          approximated }
  Bvec : rvector; {the least squares solution }
  inchar : char;
  i,j,k, imax, jmax : integer;
  t1, t2: real;

begin
  banner:='dr0102.pas -- driver for svd and least squares solution';
  {Test matrix from CNM pg 34}
  nRow:=4;
  nCol:=3;
  {Read in matrix the hard way!}
  A[1,1]:=5; A[1,2]:=1.0E-6; A[1,3]:=1; Y[1]:=1;
  A[2,1]:=6; A[2,2]:=0.999999; A[2,3]:=1; Y[2]:=2;
  A[3,1]:=7; A[3,2]:=2.00001; A[3,3]:=1; Y[3]:=3;
  A[4,1]:=8; A[4,2]:=2.9999; A[4,3]:=1; Y[4]:=4;

  Matcopy(nRow,nCol, A, W); {The matrix A is copied into working array W.}
  NashSVD( nRow, nCol, W, Z); {The singular value decomposition is
        computed for matrix A by columnwise orthogonalization of the
        working array W, to which a unit matrix of order nCol is added
        in order to form the matrix V in the bottom nCol rows of W.}
  begin
    for j:=1 to nCol do
    begin
      Zsq[j] := Z[j];
      Z[j]:= sqrt(Z[j]);
      for i:=1 to nRow do U[i,j]:=W[i,j]/Z[j];
      for i:=1 to nCol do V[i,j]:=W[i+nRow,j];
    end;
    PrtSVDResults( nRow, nCol, U, V,Z);
    begin
      svdtst(A,U,V,Z,nRow,nCol,nCol);
      writeln('Reconstruction of initial matrix from Nash working form');
      t2:=0.0; {to store largest error in reconstruction}
      for i:=1 to nRow do
      begin
        for j:=1 to nCol do
        begin
```

```
          t1:=0.0;
          for k:=1 to nCol do
            t1:=t1+W[i,k]*W[j+nRow,k]; { U * S * V-transpose}
          t1:=A[i,j]-t1; {to compute the residual}
          if abs(t1)>t2 then
          begin
            t2:=abs(t1); imax:=i; jmax:=j; {to save biggest element}
          end;
        end; {loop over columns}
      end; {loop over rows}
      writeln('Largest error is ',imax,',',jmax,'=',t2);
    end; {test svd results}
  end; {print results}
  svdlss(nRow, nCol, W, Y, Zsq, A, Bvec, 1.0e-16);
end. {dr0102.pas == svd and least squares solution}
```

Example output

For some reason not yet understood, running the compiled Pascal program does not transfer the output to our Rmarkdown output, so we resort to saving the output and then listing it as we do program code.

```
fpc ../Pascal2021/dr0102.pas
mv ../Pascal2021/dr0102 ../Pascal2021/dr0102.run
# now execute it
../Pascal2021/dr0102.run > ../Pascal2021/dr0102.out
```

```
## Free Pascal Compiler version 3.0.4+dfsg-23 [2019/11/25] for x86_64
## Copyright (c) 1993-2017 by Florian Klaempfl and others
## Target OS: Linux for x86-64
## Compiling ../Pascal2021/dr0102.pas
## dr0102.pas(487,3) Note: Local variable "inchar" not used
## Linking ../Pascal2021/dr0102
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 538 lines compiled, 0.1 sec
## 1 note(s) issued
```

```
alg01.pas -- NashSVD
End of Sweep #1-  no. of rotations performed =3
End of Sweep #2-  no. of rotations performed =3
End of Sweep #3-  no. of rotations performed =1
End of Sweep #4-  no. of rotations performed =0
 Singular values and vectors:
Singular value (1) = 1.3752987437308155E+001
Principal coordinate (U):
 0.3589430 0.4465265 0.5341101 0.6216916
Principal component (V):
 0.9587864 0.2457477 0.1426069
Singular value (2) = 1.6896078122466185E+000
Principal coordinate (U):
-0.7557625-0.3171936 0.1213826 0.5598907
Principal component (V):
-0.2090249 0.9500361-0.2318187
Singular value (3) = 1.1885323302979959E-005
Principal coordinate (U):
-0.3286873 0.1117406 0.7626745-0.5457163
```

```
Principal component (V):
-0.1924506 0.1924563 0.9622491
Column orthogonality of U
Largest inner product is 1,3= 2.8635982474156663E-011
Row orthogonality of U (NOT guaranteed in svd)
Largest inner product is 2,2=-6.8751638785273139E-001

Column orthogonality of V

Largest inner product is 3,3=-1.1102230246251565E-016

Row orthogonality of V

Largest inner product is 3,3=-1.1102230246251565E-016

Reconstruction of initial matrix

Largest error is 4,1=-1.7763568394002505E-015

Reconstruction of initial matrix from Nash working form

Largest error is 4,1= 1.7763568394002505E-015

alg02.pas == svdlss

Least squares solution

 1.0000E+000  2.4766E-006 -4.0000E+000

Residuals

-9.21E-011 -2.43E-011  7.57E-011 -1.24E-010

Sum of squared residuals = 3.0174571907166908E-020
```

For some reason, we get extra line-feed characters in the output file. They are easily removed with a text editor from the output file, but their origin is unclear. JN 2021-1-20 ??

Python

Pending . . .

R

Listing

While based on Nash and Shlien (1987), the following code shows that R can be used quite easily to implement Algorithm 1. The least squares solution (Algorithm 2) is embedded in the example output.

```
Nashsvd <- function(A, MaxRank=0, cyclelimit=25, trace = 0, rotnchk=0.3) {
## Nashsvd.R -- An attempt to remove tolerances from Nash & Shlien algorithm 190327
# Partial svd by the one-sided Jacobi method of Nash & Shlien
#  Computer Journal 1987  30(3), 268-275
#  Computer Journal 1975  18(1) 74-76
  if (cyclelimit < 6) {
```

```
      warning("Nashsvd: You cannot set cyclelimit < 6 without modifying the code")
      cyclelimit <- 6 # safety in case user tries smaller
  }
  m <- dim(A)[1]
  n <- dim(A)[2]
  if (MaxRank <= 0) MaxRank <- n
  EstColRank <- n # estimated column rank
  # Note that we may simply run algorithm to completion, or fix the
  # number of columns by EstColRank. Need ?? to fix EstColRank=0 case.??
  V <- diag(nrow=n) # identity matrix in V
  if (is.null(EstColRank)) {EstColRank <- n } # Safety check on number of svs
  z <- rep(NA, n) # column norm squares -- safety setting
  keepgoing <- TRUE
  SweepCount <- 0
  while (keepgoing) { # main loop of repeating cycles of Jacobi
    RotCount <- 0
    SweepCount <- SweepCount + 1
    if (trace > 1) cat("Sweep:", SweepCount,"\n")
##    if (EstColRank == n) { EstColRank <- n - 1 } # safety
    for (jj in 1:(EstColRank-1)) { # left column indicator
       for (kk in (jj+1): n) { # right hand column
         p <- q <- r <- 0.0 #
         oldjj <- A[,jj]
         oldkk <- A[,kk]
         p <- as.numeric(crossprod(A[,jj], A[,kk]))
         q <- as.numeric(crossprod(A[,jj], A[,jj]))
         r <- as.numeric(crossprod(A[,kk], A[,kk]))
         if (trace > 2) cat(jj," ",kk,": pqr",p," ",q," ",r," ")
         z[jj]<-q
         z[kk]<-r
         if (q >= r) { # in order, so can do test of "convergence" -- change to 0.2 * abs(p) for odd ca
            if ( (as.double(z[1]+q) > as.double(z[1]) ) && (as.double(rotnchk*abs(p)+q) > as.double(q))
              RotCount <- RotCount + 1
              p <- p/q
              r <- 1 - (r/q)
              vt <- sqrt(4*p*p +r*r)
              c0 <- sqrt(0.5*(1+r/vt))
              s0 <- p/(vt*c0)
              # rotate
              cj <- A[,jj]
              ck <- A[,kk]
              A[,jj] <- c0*cj + s0*ck
              A[,kk] <- -s0*cj + c0*ck
              cj <- V[,jj]
              ck <- V[,kk]
              V[,jj] <- c0*cj + s0*ck
              V[,kk] <- -s0*cj + c0*ck
            } else {
              if (trace > 2) cat(" NO rotn ")
            }
         } else { # out of order, must rotate
            if (trace > 2) cat("|order|")
            RotCount <- RotCount + 1
```

```
            p <- p/r
            q <- (q/r) - 1.0
            vt <- sqrt(4*p*p +q*q)
            s0 <- sqrt(0.5*(1-q/vt))
            if (p < 0) { s0 <- -s0 }
            c0 <- p/(vt*s0)
            # rotate
            cj <- A[,jj]
            ck <- A[,kk]
            A[,jj] <- c0*cj + s0*ck
            A[,kk] <- -s0*cj + c0*ck
            cj <- V[,jj]
            ck <- V[,kk]
            V[,jj] <- c0*cj + s0*ck
            V[,kk] <- -s0*cj + c0*ck
         } # end q >= r test
         nup <- as.numeric(crossprod(A[,jj], A[,kk]))
#         nuq <- as.numeric(crossprod(A[,jj], A[,jj]))
#         nur <- as.numeric(crossprod(A[,kk], A[,kk]))
         if (trace > 2) cat("    new: p= ",nup," Rel:",nup*nup/z[1],"\n")
       } # end kk
    } # end jj
    if (trace > 0) {cat("End sweep ", SweepCount,"  No. rotations =",RotCount,"\n")}
    if (trace > 2) tmp <- readline("cont.?")
    while ( (EstColRank >= 3) && (as.double(sqrt(z[EstColRank])+sqrt(z[1]) == as.double(sqrt(z[1])) )))
    # ?? Why can we not use 2? Or do we need at least 2 cols
        EstColRank <- EstColRank - 1
        if (trace > 0) {cat("Reducing rank to ", EstColRank,"\n")} # ?? can do this more cleanly
    } # end while for rank estimation
    ## Here may want to adjust for MaxRank. How??
    if (MaxRank < EstColRank) {
       if (trace > 0) {
        cat("current estimate of sv[",MaxRank,"/sv[1] =",sqrt(z[MaxRank]/z[1]),"\n")
        cat("reducing rank by 1\n")
       }
       EstColRank <- EstColRank - 1
    }
    if ( SweepCount >= cyclelimit) {
         if (trace > 0) cat("Cycle limit reached\n")
         keepgoing <- FALSE
    }
    if (RotCount == 0) {
        if (trace > 1) cat("Zero rotations in cycle\n")
        keepgoing <- FALSE
    }
  } # End main cycle loop
  z <- sqrt(z)
  A <- A %*% diag(1/z)
  ans <- list( d = z, u = A, v=V, cycles=SweepCount, rotations=RotCount)
  ans
} # end partsvd()
```

Example output

```r
# test taken from dr0102.pas
A<-matrix(0, 4,3)
A[1,]<-c(5, 1e-6, 1)
A[2,]<-c(6, 0.999999, 1)
A[3,]<-c(7, 2.00001, 1)
A[4,]<-c(8, 2.9999, 1)
print(A)
```

```
##      [,1]     [,2] [,3]
## [1,]    5 0.000001    1
## [2,]    6 0.999999    1
## [3,]    7 2.000010    1
## [4,]    8 2.999900    1
```

```r
b<-c(1,2,3,4)
print(b)
```

```
## [1] 1 2 3 4
```

```r
# try the R-base svd
sA <- svd(A)
sA
```

```
## $d
## [1] 1.375299e+01 1.689608e+00 1.188532e-05
##
## $u
##            [,1]       [,2]       [,3]
## [1,] -0.3589430 -0.7557625  0.3286873
## [2,] -0.4465265 -0.3171936 -0.1117406
## [3,] -0.5341101  0.1213826 -0.7626745
## [4,] -0.6216916  0.5598907  0.5457163
##
## $v
##            [,1]       [,2]       [,3]
## [1,] -0.9587864 -0.2090249  0.1924506
## [2,] -0.2457477  0.9500361 -0.1924563
## [3,] -0.1426069 -0.2318187 -0.9622491
```

```r
yy <- t(sA$u) %*% as.matrix(b)
xx <- sA$v %*% diag(1/sA$d) %*% yy
xx
```

```
##               [,1]
## [1,]  1.000000e+00
## [2,] -9.005019e-12
## [3,] -4.000000e+00
```

```r
# Now the Nashsvd code (this is likely NOT true to 1979 code)
source("../R/Nashsvd.R")
nsvd <- Nashsvd(A)
print(nsvd)
```

```
## $d
## [1] 1.375299e+01 1.689608e+00 1.188532e-05
##
```

```
## $u
##           [,1]       [,2]       [,3]
## [1,] 0.3589430 -0.7557625 -0.3286873
## [2,] 0.4465265 -0.3171936  0.1117406
## [3,] 0.5341101  0.1213826  0.7626745
## [4,] 0.6216916  0.5598907 -0.5457163
##
## $v
##           [,1]       [,2]       [,3]
## [1,] 0.9587864 -0.2090249 -0.1924506
## [2,] 0.2457477  0.9500361  0.1924563
## [3,] 0.1426069 -0.2318187  0.9622491
##
## $cycles
## [1] 4
##
## $rotations
## [1] 0
```

```
# Note least squares solution can be done by matrix multiplication
U <- nsvd$u
V <- nsvd$v
d <- nsvd$d
di <- 1/d
di <- diag(di) # convert to full matrix -- note entry sizes
print(di)
```

```
##            [,1]      [,2]     [,3]
## [1,] 0.07271147 0.0000000     0.00
## [2,] 0.00000000 0.5918533     0.00
## [3,] 0.00000000 0.0000000 84137.38
```

```
lsol <- t(U) %*% b
lsol <- di %*% lsol
lsol <- V %*% lsol
print(lsol)
```

```
##               [,1]
## [1,]  9.999975e-01
## [2,]  2.476918e-06
## [3,] -3.999988e+00
```

```
res <- b - A %*% lsol
print(res)
```

```
##               [,1]
## [1,]  5.027934e-11
## [2,] -1.708989e-11
## [3,] -1.166609e-10
## [4,]  8.347678e-11
```

```
cat("sumsquares = ", as.numeric(crossprod(res)))
```

```
## sumsquares =  2.339822e-20
```

```
# now set smallest singular value to 0 and in pseudo-inverse
dix <- di
```

```r
dix[3,3] <- 0
lsolx <- V %*% dix %*% t(U) %*% b
# this gives a very different least squares solution
print(lsolx)
```

```
##            [,1]
## [1,]  0.2222209
## [2,]  0.7778018
## [3,] -0.1111212
```

```r
# but the residuals (in this case) are nearly 0 too
resx <- b - A %*% lsolx
cat("sumsquares = ", as.numeric(crossprod(resx)))
```

```
## sumsquares =  2.307256e-09
```

Others

Pending . . .

?? Could we f2c the Fortran and manually tweak to get a C code?

There is also a C version in

https://github.com/LuaDist/gsl/blob/master/linalg/svd.c

==

Algorithm 3 – Givens' decomposition

The Givens and Householder decompositions of a rectangular m by n matrix A ($m >= n$) both give an m by m orthogonal matrix Q and an upper-triangular n by n matrix R whose product QR is a close approximation of A. At the time Nash (1979) was being prepared, the Givens approach seemed to give a more compact program code, though neither approach is large.

In practice, if one is trying to solve linear equations

$$Ax = b$$

or linear least squares problems of the form

$$Ax \ = b$$

then the right hand side (RHS) b can be appended to the matrix A so that the resulting working matrix

$$W = [A|b]$$

is transformed during the formation of the Q matrix into

$$W_{trans} = [R|Q'b]$$

This saves us the effort of multiplying b by the transpose of Q before we back-solve for x.

In fact, m does not have to be greater than or equal to n. However, underdetermined systems of equations do raise some issues that we will not address here.

It is therefore unnecessary to store Q, which when Nash (1979) was being prepared was a potentially large matrix. There are alternative designs of the code which could save information on the plane rotations that make up Q. Such codes can then apply the rotations to a unit matrix of the right size to reconstruct Q as needed. However, these details have largely become irrelevant in an age of cheap memory chips.

Fortran

Listing

The following listing uses the Frank matrix as a test.

```
C&&& A3
C  TEST ALGORITHM 3
C  J.C. NASH   JULY 1978, APRIL 1989
      LOGICAL SAVEQ
      CHARACTER QSAVE
      INTEGER M,N,NIN,NOUT
      REAL A(10,10),Q(10,10),EPS,S,W(10,10)
      NDIM=10
C  I/O CHANNELS
      NIN=5
      NOUT=6
   1  READ(NIN,900)M,N,QSAVE
 900  FORMAT(2I5,A1)
      WRITE(NOUT,950)M,N,QSAVE
 950  FORMAT('M=',I5,'  N=',I5,'  QSAVE=',A1)
      IF(M.EQ.0.OR.N.EQ.0)STOP
      SAVEQ=.FALSE.
```

```
      IF (QSAVE .EQ. "T") SAVEQ=.TRUE.
      CALL FRANKM(M,N,A,10)
      WRITE(NOUT,952)
 952  FORMAT('INITIAL MATRIX')
      CALL OUT(A,NDIM,M,N,NOUT)
      DO 10 I=1,M
        DO 5 J=1,N
C          COPY MATRIX TO WORKING ARRAY
          W(I,J)=A(I,J)
  5     CONTINUE
 10   CONTINUE
C  IBM MACHINE PRECISION
      EPS=16.0**(-5)
      CALL A3GR(M,N,W,10,Q,EPS,SAVEQ)
      WRITE(NOUT,953)
 953  FORMAT('FULL DECOMPOSED MATRIX')
      CALL OUT(A,NDIM,M,N,NOUT)
      IF(SAVEQ)CALL A3DT(M,N,W,NDIM,Q,NOUT,A)
      GOTO 1
      END
      SUBROUTINE A3DT(M,N,W,NDIM,Q,NOUT,A)
C  TESTS GIVENS' DECOMPOSITION
C  J.C. NASH   JULY 1978, APRIL 1989
      INTEGER M,N,NDIM,NOUT,I,J,K
      REAL A(NDIM,N),Q(NDIM,M),W(NDIM,N),S,T
      WRITE(NOUT,960)
 960  FORMAT(' Q MATRIX')
      CALL OUT(Q,NDIM,M,M,NOUT)
      WRITE(NOUT,961)
 961  FORMAT(' R MATRIX (STORED IN W')
      CALL OUT(W,NDIM,M,N,NOUT)
      IF(N.LT.M)GOTO 9
      S=1.0
      DO 5 I=1,M
        S=S*W(I,I)
   5  CONTINUE
      WRITE(NOUT,963)S
 963  FORMAT(' DETERMINANT=',1PE16.8)
   9  CONTINUE
      T=0.0
      DO 20 I=1,M
        DO 15 J=1,N
          S=0.0
          DO 10 K=1,M
            S=S+Q(I,K)*W(K,J)
  10      CONTINUE
          S=S-A(I,J)
          IF(ABS(S).GT.T)T=ABS(S)
  15    CONTINUE
  20  CONTINUE
      WRITE(NOUT,962)T
 962  FORMAT(' MAX. DEVN. OF RECONSTRUCTION FROM ORIGINAL=',E16.8)
      RETURN
```

```
      END
      SUBROUTINE OUT(A,NDIM,N,NP,NOUT)
C  J.C. NASH   JULY 1978, APRIL 1989
      INTEGER NDIM,N,NOUT,I,J
      REAL A(NDIM,NP)
      DO 20 I=1,N
        WRITE(NOUT,951)I
 951    FORMAT(' ROW',I3)
        WRITE(NOUT,952)(A(I,J),J=1,NP)
 952    FORMAT(1H ,1P5E16.8)
  20  CONTINUE
      RETURN
      END
      SUBROUTINE A3GR(M,N,A,NDIM,Q,EPS,SAVEQ)
C  ALGORITHM 3  GIVENS' REDUCTION
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  M,N  =  ORDER OF MATRIX TO BE DECOMPOSED
C  A    =  ARRAY CONTAINING MATRIX TO BE DECOMPOSED
C  NDIM   =  FIRST DIMENSION OF MATRICES - NDIM.GE.M
C  Q    =  ARRAY CONTAINING ORTHOGONAL MATRIX OF ACCUMULATED ROTATIONS
C  EPS  =  MACHINE PRECISION  = SMALLEST NO.GT.0.0 S.T. 1.0+EPS.GT.1.0
C  SAVEQ=  LOGICAL FLAG SET .TRUE. IF Q TO BE FORMED
C  STEP 0
      LOGICAL SAVEQ
      INTEGER N,M,NA,MN,I,J,K,J1
      REAL A(NDIM,N),Q(NDIM,M),EPS,TOL,B,P,S,C
      MN=M
      IF(M.GT.N)MN=N
      IF(.NOT.SAVEQ)GOTO 9
      DO 5 I=1,M
        DO 4 J=1,M
          Q(I,J)=0.0
   4    CONTINUE
        Q(I,I)=1.0
   5  CONTINUE
   9  TOL=EPS*EPS
C  STEP 1
      IF(M.EQ.1)RETURN
      DO 100 J=1,MN
        J1=J+1
        IF(J1.GT.M)GOTO 100
C  STEP 2
        DO 90 K=J1,M
C  STEP 3
          C=A(J,J)
          S=A(K,J)
          B=ABS(C)
          IF(ABS(S).GT.B)B=ABS(S)
          IF(B.EQ.0.0)GOTO 90
          C=C/B
          S=S/B
          P=SQRT(C*C+S*S)
C  STEP 4
```

```
          S=S/P
C   STEP 5
          IF(ABS(S).LT.TOL)GOTO 90
C   STEP 6
          C=C/P
C   STEP 7
          DO 75 I=1,N
            P=A(J,I)
            A(J,I)=C*P+S*A(K,I)
            A(K,I)=-S*P+C*A(K,I)
  75      CONTINUE
C   STEP 8
          IF(.NOT.SAVEQ)GOTO 90
          DO 85 I=1,M
            P=Q(I,J)
            Q(I,J)=C*P+S*Q(I,K)
            Q(I,K)=-S*P+C*Q(I,K)
  85      CONTINUE
C   STEP 9
  90    CONTINUE
C   STEP 10
 100  CONTINUE
      RETURN
      END
      SUBROUTINE FRANKM(M,N,A,NA)
C   J.C. NASH   JULY 1978, APRIL 1989
      INTEGER M,N,NA,I,J
C   INPUTS FRANK MATRIX M BY N INTO A
      REAL A(NA,N)
      DO 20 I=1,M
        DO 10 J=1,N
          A(I,J)=AMIN0(I,J)
  10    CONTINUE
  20  CONTINUE
      RETURN
      END
```

Example output

As a precaution, we use a 1 by 1 matrix as our first test. We have seen situations where otherwise reliable programs have failed on such trivial cases.

```
gfortran ../fortran/a3.f
mv ./a.out ../fortran/a3.run
../fortran/a3.run < ../fortran/a3data.in > ../fortran/a3out.txt
```

```
M=    1  N=    1  QSAVE=T
INITIAL MATRIX
 ROW  1
   1.00000000E+00
FULL DECOMPOSED MATRIX
 ROW  1
   1.00000000E+00
 Q MATRIX
 ROW  1
```

```
   1.00000000E+00
 R MATRIX (STORED IN W
 ROW  1
   1.00000000E+00
 DETERMINANT=  1.00000000E+00
 MAX. DEVN. OF RECONSTRUCTION FROM ORIGINAL=  0.00000000E+00
M=    5  N=    3  QSAVE=T
INITIAL MATRIX
 ROW  1
   1.00000000E+00  1.00000000E+00  1.00000000E+00
 ROW  2
   1.00000000E+00  2.00000000E+00  2.00000000E+00
 ROW  3
   1.00000000E+00  2.00000000E+00  3.00000000E+00
 ROW  4
   1.00000000E+00  2.00000000E+00  3.00000000E+00
 ROW  5
   1.00000000E+00  2.00000000E+00  3.00000000E+00
FULL DECOMPOSED MATRIX
 ROW  1
   1.00000000E+00  1.00000000E+00  1.00000000E+00
 ROW  2
   1.00000000E+00  2.00000000E+00  2.00000000E+00
 ROW  3
   1.00000000E+00  2.00000000E+00  3.00000000E+00
 ROW  4
   1.00000000E+00  2.00000000E+00  3.00000000E+00
 ROW  5
   1.00000000E+00  2.00000000E+00  3.00000000E+00
 Q MATRIX
 ROW  1
   4.47213590E-01 -8.94427240E-01  9.95453036E-08  1.14146687E-07 -1.93894891E-08
 ROW  2
   4.47213590E-01  2.23606765E-01 -8.66025507E-01  0.00000000E+00 -1.19209290E-07
 ROW  3
   4.47213590E-01  2.23606795E-01  2.88675159E-01 -7.07106888E-01 -4.08248186E-01
 ROW  4
   4.47213590E-01  2.23606944E-01  2.88675249E-01  7.07106769E-01 -4.08248246E-01
 ROW  5
   4.47213590E-01  2.23606795E-01  2.88674951E-01  0.00000000E+00  8.16496611E-01
 R MATRIX (STORED IN W
 ROW  1
   2.23606801E+00  4.02492237E+00  5.36656284E+00
 ROW  2
   1.92373264E-08  8.94427299E-01  1.56524777E+00
 ROW  3
   2.48352734E-08  1.40489522E-08  8.66025269E-01
 ROW  4
   4.86669869E-08  2.58095696E-08  0.00000000E+00
 ROW  5
  -1.40489469E-08 -4.96705121E-09  0.00000000E+00
 MAX. DEVN. OF RECONSTRUCTION FROM ORIGINAL=  0.29802322E-06
M=    0  N=    0  QSAVE=
```

BASIC

Listing

The following listing also uses the Frank matrix as a test. The code has been adjusted for fixed input to allow it to be run within the `knitr` processor for `Rmarkdown`.

```
2 REM DIM A(10,10),Q(10,10)
10 PRINT "TEST GIVENS - GIFT - ALG 3"
12 LET M8=10
14 LET N8=10
20 DIM A(M8,N8),Q(M8,M8)
25 REM PRINT "M=",
30 REM INPUT M
32 LET M=5
40 REM PRINT "  N=",
50 REM INPUT N
52 LET N=3
70 GOSUB 1500
80 PRINT "ORIGINAL",
85 GOSUB 790
90 GOSUB 500 : REM GIVENS DECOMPOSITION
94 PRINT "FINAL ";
96 GOSUB 790
97 PRINT "FINAL ";
98 GOSUB 840
100 PRINT "RECOMBINATION "
110 FOR I=1 TO M
111   PRINT "ROW";I;":";
120   FOR J=1 TO N
130     LET S=0
140     FOR K=1 TO M
150       LET S=S+Q(I,K)*A(K,J)
160     NEXT K
170     PRINT S;" ";
210   NEXT J
220   PRINT
230 NEXT I
240 QUIT
245 REM STOP
500 REM GIVENS TRIANGULARIZATION
520 PRINT "GIVENS TRIANGULARIZATION DEC 12 77"
540 FOR I=1 TO M
545   FOR J=1 TO M
550     LET Q(I,J)=0
555   NEXT J
560   LET Q(I,I)=1
565 NEXT I
575 REM GOSUB 840: REM PRINT ORIGINAL Q MATRIX
580 LET E1=1E-7 : REM NORTH STAR 8 DIGIT -- can be changed!
585 LET T9=E1*E1
600 FOR J=1 TO N-1
605   FOR K=J+1 TO M
610     LET C=A(J,J)
615     LET S=A(K,J)
```

```
625     REM PRINT "J=",J,"  K=",K,"  A[J,J]=",C,"  A[K,J]=",S
630     REM PRINT "BYPASS SAFETY DIVISION  ",
635     REM GOTO 660
640     LET B=ABS(C)
645     IF ABS(S)<=B THEN GOTO 655
650     LET B=ABS(S)
655     LET C=C/B
660     LET S=S/B
665     IF B=0 THEN GOTO 770
670     LET P=SQR(C*C+S*S)
680     LET S=S/P
685     IF ABS(S)<T9 THEN GOTO 770
690     LET C=C/P
695     FOR I=1 TO N
700       LET P=A(J,I)
705       LET A(J,I)=C*P+S*A(K,I)
710       LET A(K,I)=-S*P+C*A(K,I)
715      NEXT I
720     IF J=N-1 THEN GOTO 730
730     REM IF I5=0 THEN GOTO 770
735     FOR I=1 TO M
740       LET P=Q(I,J)
745       LET Q(I,J)=C*P+S*Q(I,K)
750       LET Q(I,K)=-S*P+C*Q(I,K)
755     NEXT I
770     REM Possible print point
775   NEXT K
780 NEXT J
785 RETURN
790 PRINT "  A MATRIX"
795 FOR H=1 TO M
800   PRINT "ROW";H;":";
805   FOR L=1 TO N
810      PRINT A(H,L);" ";
815   NEXT L
820   PRINT
825 NEXT H
830 PRINT
835 RETURN
840 PRINT "  Q MATRIX"
845 FOR H=1 TO M
850   PRINT "ROW";H;":";
855   FOR L=1 TO M
860     PRINT Q(H,L);" ";
865   NEXT L
870   PRINT
875 NEXT H
880 PRINT
885 RETURN
1500 REM PREPARE FRANK MATRIX IN A
1510 FOR I=1 TO M
1530 FOR J=1 TO N
1540 IF (I <= J) THEN LET A(I,J)=I ELSE LET A(I,J)=J
```

```
1550 NEXT J
1560 NEXT I
1570 RETURN
1600 END
```

Example output

As a precaution, we use a 1 by 1 matrix as our first test. We have seen situations where otherwise reliable programs have failed on such trivial cases.

```
bwbasic ../BASIC/a3.bas
```

```
## Bywater BASIC Interpreter/Shell, version 2.20 patch level 2
## Copyright (c) 1993, Ted A. Campbell
## Copyright (c) 1995-1997, Jon B. Volkoff
##
## TEST GIVENS - GIFT - ALG 3
## ORIGINAL
##    A MATRIX
## ROW 1: 1   1   1
## ROW 2: 1   2   2
## ROW 3: 1   2   3
## ROW 4: 1   2   3
## ROW 5: 1   2   3
##
## GIVENS TRIANGULARIZATION DEC 12 77
## FINAL    A MATRIX
## ROW 1: 2.2360680  4.0249224  5.3665631
## ROW 2: 0  0.8944272  1.5652476
## ROW 3: 0  0  0.7071068
## ROW 4: 0  0  0.4082483
## ROW 5: -0  -0  0.2886751
##
## FINAL    Q MATRIX
## ROW 1: 0.4472136  -0.8944272  0  0  0
## ROW 2: 0.4472136  0.2236068  -0.7071068  -0.4082483  -0.2886751
## ROW 3: 0.4472136  0.2236068  0.7071068  -0.4082483  -0.2886751
## ROW 4: 0.4472136  0.2236068  0  0.8164966  -0.2886751
## ROW 5: 0.4472136  0.2236068  0  0  0.8660254
##
## RECOMBINATION
## ROW 1: 1   1   1
## ROW 2: 1   2   2.0000000
## ROW 3: 1   2   3
## ROW 4: 1.0000000  2.0000000  3.0000000
## ROW 5: 1.0000000  2.0000000  3.0000000
```

Pascal

Listing – column-wise approach

```
program givrun(input, output);
{dr03.PAS ==  driver for Givens' reduction of a matrix

          Copyright 1988 J.C.Nash
```

```
}
{I constype.def}
{constype.def ==
  This file contains various definitions and type statements which are
  used throughout the collection of "Compact Numerical Methods".  In many
  cases not all definitions are needed, and users with very tight memory
  constraints may wish to remove some of the lines of this file when
  compiling certain programs.

  Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}
{uses Dos, Crt;} {Turbo Pascal 5.0 Modules}
{ 1. Interrupt, Unit, Interface, Implementation, Uses are reserved words now.}
{ 2. System,Dos,Crt are standard unit names in Turbo 5.0.}

const
  big = 1.0E+35;    {a very large number}
  Maxconst = 25;    {Maximum number of constants in data record}
  Maxobs = 100;     {Maximum number of observations in data record}
  Maxparm  = 25;    {Maximum number of parameters to adjust}
  Maxvars = 10;     {Maximum number of variables in data record}
  acctol = 0.0001;  {acceptable point tolerance for minimisation codes}
  maxm = 20;        {Maximum number or rows in a matrix}
  maxn = 20;        {Maximum number of columns in a matrix}
  maxmn = 40;       {maxn+maxm, the number of rows in a working array}
  maxsym = 210;     {maximum number of elements of a symmetric matrix
              which need to be stored = maxm * (maxm + 1)/2 }
  reltest = 10.0;   {a relative size used to check equality of numbers.
              Numbers x and y are considered equal if the
              floating-point representation of reltest+x equals
              that of reltest+y.}
  stepredn = 0.2;   {factor to reduce stepsize in line search}
  yearwrit = 1990;  {year in which file was written}

type
  str2 = string[2];
  rmatrix = array[1..maxm, 1..maxn] of real; {a real matrix}
  wmatrix = array[1..maxmn, 1..maxn] of real; {a working array, formed
                  as one real matrix stacked on another}
  smatvec = array[1..maxsym] of real; {a vector to store a symmetric matrix
              as the row-wise expansion of its lower triangle}
  rvector = array[1..maxm] of real;  {a real vector. We will use vectors
              of m elements always. While this is NOT space efficient,
              it simplifies program codes.}
  cgmethodtype= (Fletcher_Reeves,Polak_Ribiere,Beale_Sorenson);
    {three possible forms of the conjugate gradients updating formulae}
  probdata = record
          m     : integer; {number of observations}
          nvar  : integer; {number of variables}
          nconst: integer; {number of constants}
          vconst: array[1..Maxconst] of real;
```

```
          Ydata : array[1..Maxobs, 1..Maxvars] of real;
          nlls  : boolean; {true if problem is nonlinear least squares}
        end;
{
  NOTE: Pascal does not let us define the work-space for the function
  within the user-defined code.  This is a weakness of Pascal for this
  type of work.
}
var {global definitions}
  banner     : string[80]; {program name and description}

function calceps:real;
{calceps.pas ==
  This function returns the machine EPSILON or floating point tolerance,
  the smallest positive real number such that 1.0 + EPSILON > 1.0.
  EPSILON is needed to set various tolerances for different algorithms.
  While it could be entered as a constant, I prefer to calculate it, since
  users tend to move software between machines without paying attention to
  the computing environment. Note that more complete routines exist.
}
var
  e,e0: real;
  i: integer;
begin {calculate machine epsilon}
  e0 := 1; i:=0;
  repeat
    e0 := e0/2; e := 1+e0;  i := i+1;
  until (e=1.0) or (i=50); {note safety check}
  e0 := e0*2;
{ Writeln('Machine EPSILON =',e0);}
  calceps:=e0;
end; {calceps}



procedure givens( nRow,nCol : integer;
                  var A, Q: rmatrix);
var
 i, j, k, mn: integer;
 b, c, eps, p, s : real;

begin
  writeln('alg03.pas -- Givens',chr(39),' reduction -- column-wise');
  mn  :=  nRow; if nRow>nCol then mn  :=  nCol;
  for i  :=  1 to nRow do
  begin
    for j := 1 to nRow do Q[i,j] := 0.0;
    Q[i,i] := 1.0;
  end;
  eps := calceps;
  for j := 1 to (mn-1) do
  begin
    for k := (j+1) to nRow do
```

```
    begin
      c := A[j,j]; s := A[k,j];
      b := abs(c); if abs(s)>b then b := abs(s);
      if b>0 then
      begin
        c := c/b; s := s/b;
        p := sqrt(c*c+s*s);
        s := s/p;
        if abs(s)>=eps then
        begin
          c := c/p;
          for i := 1 to nCol do
          begin
            p := A[j,i]; A[j,i] := c*p+s*A[k,i]; A[k,i] := -s*p+c*A[k,i];
          end;
          for i := 1 to nRow do
          begin
            p := Q[i,j]; Q[i,j] := c*p+s*Q[i,k]; Q[i,k] := -s*p+c*Q[i,k];
          end;
        end;
      end;
    end;
  end;
end;

Procedure Frank2(var m, n: integer; var A: rmatrix);
var
  i,j: integer;
begin
    for i:=1 to m do
    begin
        for j:=1 to n do
        begin
          write(i,' ',j,';');
          if (i <= j) then
             A[i,j]:=i
          else
             A[i,j]:=j;
          writeln(A[i,j]);
        end;
    end;
end;



var
  A, Q: rmatrix;
  i, j, k, nRow, nCol : integer;
  Acopy : rmatrix;
  s : real;

begin
  banner:='dr03.pas -- driver for Givens'+chr(39)+' reduction';
```

```
nRow := 5;
nCol := 3; {Specific to this example.}
writeln('Size of problem (rows, columns)  (',nRow,', ',nCol,')');
writeln('Frank matrix example');
Frank2(nRow, nCol, A);
writeln('Matrix A');
for i:=1 to nRow do
begin
  for j:=1 to nCol do
  begin
    Acopy[i,j]:=A[i,j];
    write(A[i,j]:10:5,' ');
    if (7 * (j div 7) = j) and (j<nCol) then
    begin
      writeln;
    end;
  end;
  writeln;
end;
givens(nRow,nCol,A,Q);
writeln('Decomposition');
writeln('Q');
for i:=1 to nRow do
begin
  for j:=1 to nRow do
  begin
    write(Q[i,j]:10:5,' ');
    if (7 * (j div 7) = j) and (j<nRow) then
    begin
      writeln;
    end;
  end;
  writeln;
end;
writeln('R');
for i:=1 to nRow do
begin
  for j:=1 to nCol do
  begin
    write(A[i,j]:10:5,' ');
    if (7 * (j div 7) = j) and (j<nCol) then
    begin
      writeln;
    end;
  end;
  writeln;
end;
writeln('Q*R - Acopy');
for i:=1 to nRow do
begin
  for j:=1 to nCol do
  begin
    s:=-Acopy[i,j];
```

```
      for k:=1 to nRow do s:=s+Q[i,k]*A[k,j];
      write(s:10,' ');
      if (7 * (j div 7) = j) and (j<nRow) then
      begin
        writeln;
      end;
    end;
    writeln;
  end;
end. {dr03.pas == Givens' reduction driver}
```

Example output – column-wise approach

```
fpc ../Pascal2021/dr03.pas
mv ../Pascal2021/dr03 ../Pascal2021/dr03.run
../Pascal2021/dr03.run >../Pascal2021/dr03.out
```

```
## Free Pascal Compiler version 3.0.4+dfsg-23 [2019/11/25] for x86_64
## Copyright (c) 1993-2017 by Florian Klaempfl and others
## Target OS: Linux for x86-64
## Compiling ../Pascal2021/dr03.pas
## Linking ../Pascal2021/dr03
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 226 lines compiled, 0.1 sec
```

```
Size of problem (rows, columns)  (5, 3)
Frank matrix example
1 1; 1.0000000000000000E+000
1 2; 1.0000000000000000E+000
1 3; 1.0000000000000000E+000
2 1; 1.0000000000000000E+000
2 2; 2.0000000000000000E+000
2 3; 2.0000000000000000E+000
3 1; 1.0000000000000000E+000
3 2; 2.0000000000000000E+000
3 3; 3.0000000000000000E+000
4 1; 1.0000000000000000E+000
4 2; 2.0000000000000000E+000
4 3; 3.0000000000000000E+000
5 1; 1.0000000000000000E+000
5 2; 2.0000000000000000E+000
5 3; 3.0000000000000000E+000
Matrix A
   1.00000    1.00000    1.00000
   1.00000    2.00000    2.00000
   1.00000    2.00000    3.00000
   1.00000    2.00000    3.00000
   1.00000    2.00000    3.00000
alg03.pas -- Givens' reduction -- column-wise
Decomposition
Q
   0.44721   -0.89443    0.00000    0.00000    0.00000
   0.44721    0.22361   -0.70711   -0.40825   -0.28868
   0.44721    0.22361    0.70711   -0.40825   -0.28868
```

```
   0.44721    0.22361    0.00000    0.81650   -0.28868
   0.44721    0.22361    0.00000    0.00000    0.86603
R
   2.23607    4.02492    5.36656
   0.00000    0.89443    1.56525
   0.00000    0.00000    0.70711
   0.00000    0.00000    0.40825
  -0.00000   -0.00000    0.28868
Q*R - Acopy
 1.45E-016  2.22E-016  6.95E-016
 1.45E-016 -1.03E-016 -1.11E-016
 2.81E-016  2.86E-016  2.36E-016
-1.26E-016 -4.64E-016 -8.05E-016
-2.22E-016 -2.46E-016 -5.00E-016
```

Algorithms 5 and 6 – Gaussian elimination and back-solution

Fortran

```
gfortran ../fortran/dr0506.f
mv ./a.out ../fortran/dr0506.run
../fortran/dr0506.run > ../fortran/dr0506out.txt
```

```
 ORDER=   4  ORIGINAL MATRIX WITH RHS APPENDED
 ROW  1
  1.00000E+00 1.00000E+00 1.00000E+00 1.00000E+00 4.00000E+00
 ROW  2
  1.00000E+00 2.00000E+00 2.00000E+00 2.00000E+00 7.00000E+00
 ROW  3
  1.00000E+00 2.00000E+00 3.00000E+00 3.00000E+00 9.00000E+00
 ROW  4
  1.00000E+00 2.00000E+00 3.00000E+00 4.00000E+00 1.00000E+01
 DETERMINANT= 1.00000E+00
 SOLN X(   1)= 1.00000E+00   ERROR= 0.00000E+00
 SOLN X(   2)= 1.00000E+00   ERROR= 0.00000E+00
 SOLN X(   3)= 1.00000E+00   ERROR= 0.00000E+00
 SOLN X(   4)= 1.00000E+00   ERROR= 0.00000E+00
 ORDER=   8  ORIGINAL MATRIX WITH RHS APPENDED
 ROW  1
  1.00000E+00 1.00000E+00 1.00000E+00 1.00000E+00 1.00000E+00
  1.00000E+00 1.00000E+00 1.00000E+00 8.00000E+00
 ROW  2
  1.00000E+00 2.00000E+00 2.00000E+00 2.00000E+00 2.00000E+00
  2.00000E+00 2.00000E+00 2.00000E+00 1.50000E+01
 ROW  3
  1.00000E+00 2.00000E+00 3.00000E+00 3.00000E+00 3.00000E+00
  3.00000E+00 3.00000E+00 3.00000E+00 2.10000E+01
 ROW  4
  1.00000E+00 2.00000E+00 3.00000E+00 4.00000E+00 4.00000E+00
  4.00000E+00 4.00000E+00 4.00000E+00 2.60000E+01
 ROW  5
  1.00000E+00 2.00000E+00 3.00000E+00 4.00000E+00 5.00000E+00
  5.00000E+00 5.00000E+00 5.00000E+00 3.00000E+01
 ROW  6
  1.00000E+00 2.00000E+00 3.00000E+00 4.00000E+00 5.00000E+00
  6.00000E+00 6.00000E+00 6.00000E+00 3.30000E+01
 ROW  7
  1.00000E+00 2.00000E+00 3.00000E+00 4.00000E+00 5.00000E+00
  6.00000E+00 7.00000E+00 7.00000E+00 3.50000E+01
 ROW  8
  1.00000E+00 2.00000E+00 3.00000E+00 4.00000E+00 5.00000E+00
  6.00000E+00 7.00000E+00 8.00000E+00 3.60000E+01
 DETERMINANT= 1.00000E+00
 SOLN X(   1)= 1.00000E+00   ERROR= 0.00000E+00
 SOLN X(   2)= 1.00000E+00   ERROR= 0.00000E+00
 SOLN X(   3)= 1.00000E+00   ERROR= 0.00000E+00
 SOLN X(   4)= 1.00000E+00   ERROR= 0.00000E+00
 SOLN X(   5)= 1.00000E+00   ERROR= 0.00000E+00
 SOLN X(   6)= 1.00000E+00   ERROR= 0.00000E+00
```

```
 SOLN X(  7)= 1.00000E+00   ERROR= 0.00000E+00
 SOLN X(  8)= 1.00000E+00   ERROR= 0.00000E+00
```

Pascal

Listing – column-wise approach

```
program givrun(input, output);
{dr03.PAS ==  driver for Givens' reduction of a matrix

          Copyright 1988 J.C.Nash
}
{I constype.def}
{constype.def ==
  This file contains various definitions and type statements which are
  used throughout the collection of "Compact Numerical Methods".  In many
  cases not all definitions are needed, and users with very tight memory
  constraints may wish to remove some of the lines of this file when
  compiling certain programs.

  Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}
{uses Dos, Crt;} {Turbo Pascal 5.0 Modules}
{ 1. Interrupt, Unit, Interface, Implementation, Uses are reserved words now.}
{ 2. System,Dos,Crt are standard unit names in Turbo 5.0.}

const
  big = 1.0E+35;    {a very large number}
  Maxconst = 25;    {Maximum number of constants in data record}
  Maxobs = 100;     {Maximum number of observations in data record}
  Maxparm  = 25;    {Maximum number of parameters to adjust}
  Maxvars = 10;     {Maximum number of variables in data record}
  acctol = 0.0001;  {acceptable point tolerance for minimisation codes}
  maxm = 20;        {Maximum number or rows in a matrix}
  maxn = 20;        {Maximum number of columns in a matrix}
  maxmn = 40;       {maxn+maxm, the number of rows in a working array}
  maxsym = 210;     {maximum number of elements of a symmetric matrix
              which need to be stored = maxm * (maxm + 1)/2 }
  reltest = 10.0;   {a relative size used to check equality of numbers.
              Numbers x and y are considered equal if the
              floating-point representation of reltest+x equals
              that of reltest+y.}
  stepredn = 0.2;   {factor to reduce stepsize in line search}
  yearwrit = 1990;  {year in which file was written}

type
  str2 = string[2];
  rmatrix = array[1..maxm, 1..maxn] of real; {a real matrix}
  wmatrix = array[1..maxmn, 1..maxn] of real; {a working array, formed
                  as one real matrix stacked on another}
  smatvec = array[1..maxsym] of real; {a vector to store a symmetric matrix
```

```
              as the row-wise expansion of its lower triangle}
  rvector = array[1..maxm] of real;  {a real vector. We will use vectors
              of m elements always. While this is NOT space efficient,
              it simplifies program codes.}
  cgmethodtype= (Fletcher_Reeves,Polak_Ribiere,Beale_Sorenson);
    {three possible forms of the conjugate gradients updating formulae}
  probdata = record
          m     : integer; {number of observations}
          nvar  : integer; {number of variables}
          nconst: integer; {number of constants}
          vconst: array[1..Maxconst] of real;
          Ydata : array[1..Maxobs, 1..Maxvars] of real;
          nlls  : boolean; {true if problem is nonlinear least squares}
        end;
{
  NOTE: Pascal does not let us define the work-space for the function
  within the user-defined code.  This is a weakness of Pascal for this
  type of work.
}
var {global definitions}
  banner     : string[80]; {program name and description}

function calceps:real;
{calceps.pas ==
  This function returns the machine EPSILON or floating point tolerance,
  the smallest positive real number such that 1.0 + EPSILON > 1.0.
  EPSILON is needed to set various tolerances for different algorithms.
  While it could be entered as a constant, I prefer to calculate it, since
  users tend to move software between machines without paying attention to
  the computing environment. Note that more complete routines exist.
}
var
  e,e0: real;
  i: integer;
begin {calculate machine epsilon}
  e0 := 1; i:=0;
  repeat
    e0 := e0/2; e := 1+e0;  i := i+1;
  until (e=1.0) or (i=50); {note safety check}
  e0 := e0*2;
{ Writeln('Machine EPSILON =',e0);}
  calceps:=e0;
end; {calceps}



procedure givens( nRow,nCol : integer;
                  var A, Q: rmatrix);
var
 i, j, k, mn: integer;
 b, c, eps, p, s : real;

begin
```

```
  writeln('alg03.pas -- Givens',chr(39),' reduction -- column-wise');
  mn  :=  nRow; if nRow>nCol then mn  :=  nCol;
  for i  :=  1 to nRow do
  begin
    for j := 1 to nRow do Q[i,j] := 0.0;
    Q[i,i] := 1.0;
  end;
  eps := calceps;
  for j := 1 to (mn-1) do
  begin
    for k := (j+1) to nRow do
    begin
      c := A[j,j]; s := A[k,j];
      b := abs(c); if abs(s)>b then b := abs(s);
      if b>0 then
      begin
        c := c/b; s := s/b;
        p := sqrt(c*c+s*s);
        s := s/p;
        if abs(s)>=eps then
        begin
          c := c/p;
          for i := 1 to nCol do
          begin
            p := A[j,i]; A[j,i] := c*p+s*A[k,i]; A[k,i] := -s*p+c*A[k,i];
          end;
          for i := 1 to nRow do
          begin
            p := Q[i,j]; Q[i,j] := c*p+s*Q[i,k]; Q[i,k] := -s*p+c*Q[i,k];
          end;
        end;
      end;
    end;
  end;
end;

Procedure Frank2(var m, n: integer; var A: rmatrix);
var
  i,j: integer;
begin
    for i:=1 to m do
    begin
        for j:=1 to n do
        begin
          write(i,' ',j,';');
          if (i <= j) then
             A[i,j]:=i
          else
             A[i,j]:=j;
          writeln(A[i,j]);
        end;
    end;
end;
```

```
var
  A, Q: rmatrix;
  i, j, k, nRow, nCol : integer;
  Acopy : rmatrix;
  s : real;

begin
  banner:='dr03.pas -- driver for Givens'+chr(39)+' reduction';
  nRow := 5;
  nCol := 3; {Specific to this example.}
  writeln('Size of problem (rows, columns)  (',nRow,', ',nCol,')');
  writeln('Frank matrix example');
  Frank2(nRow, nCol, A);
  writeln('Matrix A');
  for i:=1 to nRow do
  begin
    for j:=1 to nCol do
    begin
      Acopy[i,j]:=A[i,j];
      write(A[i,j]:10:5,' ');
      if (7 * (j div 7) = j) and (j<nCol) then
      begin
        writeln;
      end;
    end;
    writeln;
  end;
  givens(nRow,nCol,A,Q);
  writeln('Decomposition');
  writeln('Q');
  for i:=1 to nRow do
  begin
    for j:=1 to nRow do
    begin
      write(Q[i,j]:10:5,' ');
      if (7 * (j div 7) = j) and (j<nRow) then
      begin
        writeln;
      end;
    end;
    writeln;
  end;
  writeln('R');
  for i:=1 to nRow do
  begin
    for j:=1 to nCol do
    begin
      write(A[i,j]:10:5,' ');
      if (7 * (j div 7) = j) and (j<nCol) then
      begin
        writeln;
```

```
      end;
    end;
    writeln;
  end;
  writeln('Q*R - Acopy');
  for i:=1 to nRow do
  begin
    for j:=1 to nCol do
    begin
      s:=-Acopy[i,j];
      for k:=1 to nRow do s:=s+Q[i,k]*A[k,j];
      write(s:10,' ');
      if (7 * (j div 7) = j) and (j<nRow) then
      begin
        writeln;
      end;
    end;
    writeln;
  end;
end. {dr03.pas == Givens' reduction driver}
```

Example output – column-wise approach

```
fpc ../Pascal2021/dr0506.pas
mv ../Pascal2021/dr0506 ../Pascal2021/dr0506.run
../Pascal2021/dr0506.run >../Pascal2021/dr0506.out
```

```
## Free Pascal Compiler version 3.0.4+dfsg-23 [2019/11/25] for x86_64
## Copyright (c) 1993-2017 by Florian Klaempfl and others
## Target OS: Linux for x86-64
## Compiling ../Pascal2021/dr0506.pas
## Linking ../Pascal2021/dr0506
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 257 lines compiled, 0.1 sec
```

```
Data matrix :4 by 5
Row 1
   1.00000    2.00000    3.00000    4.00000   10.00000
Row 2
   2.00000    2.00000    3.00000    4.00000   11.00000
Row 3
   3.00000    3.00000    3.00000    4.00000   13.00000
Row 4
   4.00000    4.00000    4.00000    4.00000   16.00000

tol for pivod =  2.8421709430404007E-014
alg05.pas -- Gauss elimination with partial pivoting
Interchanging rows 4 and 1
Interchanging rows 4 and 2
Interchanging rows 4 and 3
Gauss elimination complete -- determinant = -2.4000000000000000E+001
returned matrix 4 by 5
Row 1
   4.00000    4.00000    4.00000    4.00000   16.00000 Row 2
```

```
   0.50000    1.00000    2.00000    3.00000    6.00000 Row 3
   0.75000    0.00000    1.00000    2.00000    3.00000 Row 4
   0.25000    0.00000    0.00000    1.00000    1.00000
alg06.pas -- Gauss elimination back-substitution
Solution 1
   1.00000    1.00000    1.00000    1.00000
Residuals
 0.00E+000  0.00E+000  0.00E+000  0.00E+000
Sum of squared residuals =  0.0000000000000000E+000
Data matrix :8 by 9
Row 1
   1.00000    2.00000    3.00000    4.00000    5.00000    6.00000    7.00000
   8.00000   36.00000
Row 2
   2.00000    2.00000    3.00000    4.00000    5.00000    6.00000    7.00000
   8.00000   37.00000
Row 3
   3.00000    3.00000    3.00000    4.00000    5.00000    6.00000    7.00000
   8.00000   39.00000
Row 4
   4.00000    4.00000    4.00000    4.00000    5.00000    6.00000    7.00000
   8.00000   42.00000
Row 5
   5.00000    5.00000    5.00000    5.00000    5.00000    6.00000    7.00000
   8.00000   46.00000
Row 6
   6.00000    6.00000    6.00000    6.00000    6.00000    6.00000    7.00000
   8.00000   51.00000
Row 7
   7.00000    7.00000    7.00000    7.00000    7.00000    7.00000    7.00000
   8.00000   57.00000
Row 8
   8.00000    8.00000    8.00000    8.00000    8.00000    8.00000    8.00000
   8.00000   64.00000

tol for pivod =  1.1368683772161603E-013
alg05.pas -- Gauss elimination with partial pivoting
Interchanging rows 8 and 1
Interchanging rows 8 and 2
Interchanging rows 8 and 3
Interchanging rows 8 and 4
Interchanging rows 8 and 5
Interchanging rows 8 and 6
Interchanging rows 8 and 7
Gauss elimination complete -- determinant = -4.0320000000000000E+004
returned matrix 8 by 9
Row 1
   8.00000    8.00000    8.00000    8.00000    8.00000    8.00000    8.00000
   8.00000   64.00000 Row 2
   0.25000    1.00000    2.00000    3.00000    4.00000    5.00000    6.00000
   7.00000   28.00000 Row 3
   0.37500    0.00000    1.00000    2.00000    3.00000    4.00000    5.00000
   6.00000   21.00000 Row 4
```

```
   0.50000    0.00000    0.00000    1.00000    2.00000    3.00000    4.00000
   5.00000   15.00000 Row 5
   0.62500    0.00000    0.00000    0.00000    1.00000    2.00000    3.00000
   4.00000   10.00000 Row 6
   0.75000    0.00000    0.00000    0.00000    0.00000    1.00000    2.00000
   3.00000    6.00000 Row 7
   0.87500    0.00000    0.00000    0.00000    0.00000    0.00000    1.00000
   2.00000    3.00000 Row 8
   0.12500    0.00000    0.00000    0.00000    0.00000    0.00000    0.00000
   1.00000    1.00000
alg06.pas -- Gauss elimination back-substitution
Solution 1
   1.00000    1.00000    1.00000    1.00000    1.00000    1.00000    1.00000
   1.00000
Residuals
 0.00E+000  0.00E+000  0.00E+000  0.00E+000  0.00E+000  0.00E+000  0.00E+000
 0.00E+000
Sum of squared residuals =  0.0000000000000000E+000
```

Algorithms 7 and 8 – Choleski decomposition and solution

Listing

```
program dr0708(input,output);
{dr0708.pas == driver program to test procedures for Choleski (Alg07)
      and Choleski back-substitution (Alg08)

          Copyright 1988 J.C.Nash
}
{constype.def ==
  This file contains various definitions and type statements which are
  used throughout the collection of "Compact Numerical Methods".  In many
  cases not all definitions are needed, and users with very tight memory
  constraints may wish to remove some of the lines of this file when
  compiling certain programs.

  Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}

const
  big = 1.0E+35;    {a very large number}
  Maxconst = 25;    {Maximum number of constants in data record}
  Maxobs = 100;     {Maximum number of observations in data record}
  Maxparm  = 25;    {Maximum number of parameters to adjust}
  Maxvars = 10;     {Maximum number of variables in data record}
  acctol = 0.0001;  {acceptable point tolerance for minimisation codes}
  maxm = 20;        {Maximum number or rows in a matrix}
  maxn = 20;        {Maximum number of columns in a matrix}
  maxmn = 40;       {maxn+maxm, the number of rows in a working array}
  maxsym = 210;     {maximum number of elements of a symmetric matrix
              which need to be stored = maxm * (maxm + 1)/2 }
```

```
  reltest = 10.0;   {a relative size used to check equality of numbers.
              Numbers x and y are considered equal if the
              floating-point representation of reltest+x equals
              that of reltest+y.}
  stepredn = 0.2;   {factor to reduce stepsize in line search}
  yearwrit = 1990;  {year in which file was written}

type
  str2 = string[2];
  rmatrix = array[1..maxm, 1..maxn] of real; {a real matrix}
  wmatrix = array[1..maxmn, 1..maxn] of real; {a working array, formed
                  as one real matrix stacked on another}
  smatvec = array[1..maxsym] of real; {a vector to store a symmetric matrix
              as the row-wise expansion of its lower triangle}
  rvector = array[1..maxm] of real;  {a real vector. We will use vectors
              of m elements always. While this is NOT space efficient,
              it simplifies program codes.}
  cgmethodtype= (Fletcher_Reeves,Polak_Ribiere,Beale_Sorenson);
    {three possible forms of the conjugate gradients updating formulae}
  probdata = record
          m     : integer; {number of observations}
          nvar  : integer; {number of variables}
          nconst: integer; {number of constants}
          vconst: array[1..Maxconst] of real;
          Ydata : array[1..Maxobs, 1..Maxvars] of real;
          nlls  : boolean; {true if problem is nonlinear least squares}
        end;
{
  NOTE: Pascal does not let us define the work-space for the function
  within the user-defined code.  This is a weakness of Pascal for this
  type of work.

}
var {global definitions}
  banner     : string[80]; {program name and description}

Procedure matrixin(var m, n: integer; var A: rmatrix;
              var avector: smatvec; var sym :boolean);

{matrixin.pas --

  This procedure generates an m by n real matrix in both
  A or avector.

  A is of type rmatrix, an array[1..nmax, 1..nmax] of real
  where nmax >= n for all possible n to be provided.

  avector is of type rvector, an array[1..nmax*(nmax+1)/2]
  of real, with nmax as above.

  sym is set true if the resulting matrix is symmetric.
}
```

```
var
  temp : real;
  i,j,k: integer;
  inchar: char;
  mtype: integer;
  mn : integer;

begin
  if (m=0) or (n=0) then
  begin
    writeln;
    writeln('******* Matrix dimensions zero *********');
    halt;
  end;
  writeln('Matrixin.pas -- generate or input a real matrix ',m,' by ',n);
  writeln('Possible matrices to generate:');
  writeln('0) Keyboard or console file input');
  writeln('1) Hilbert segment');
  writeln('2) Ding Dong');
  writeln('3) Moler');
  writeln('4) Frank symmetric');
  writeln('5) Bordered symmetric');
  writeln('6) Diagonal');
  writeln('7) Wilkinson W+');
  writeln('8) Wilkinson W-');
  writeln('9) Constant');
  writeln('10) Unit');
{ Note: others could be added.}
  mn:=n;
  if m>mn then mn:=m; {mn is maximum of m and n}
  write('Enter type to generate ');
  readln(mtype);
  writeln(mtype);
  case mtype of
    0: begin
      sym:=false;
      if m=n then
      begin
        write('Is matrix symmetric? ');
        readln(inchar);
        writeln(inchar);
        if (inchar='y') or (inchar='Y') then sym:=true else sym:=false;
      end; {ask if symmetric}
      if sym then
      begin
        for i:=1 to n do
        begin
          writeln('Row ',i,' lower triangle elements');
          for j:=1 to i do
          begin
            read(A[i,j]);
            write(A[i,j]:10:5,' ');
            A[j,i]:=A[i,j];
            if (7*(j div 7) = j) and (j<i) then writeln;
```

```
      end;
      writeln;
    end;
  end {symmetric matrix}
  else
  begin {not symmetric}
    for i:=1 to m do
    begin
      writeln('Row ',i);
      for j:=1 to n do
      begin
        read(A[i,j]);
        write(A[i,j]:10:5,' ');
      end; {loop on j}
      writeln;
    end; {loop on i}
  end; {else not symmetric}
end; {case 0 -- input of matrix}
1: begin {Hilbert}
  for i:=1 to mn do
    for j:=1 to mn do
      A[i,j]:=1.0/(i+j-1.0);
  if m=n then sym:=true;
end;
2: begin {Ding Dong}
  for i:=1 to mn do
    for j:=1 to mn do
      A[i,j]:=0.5/(1.5+n-i-j);
  if m=n then sym:=true;
end;
3: begin {Moler}
  for i:=1 to mn do
  begin
    for j:=1 to i do
    begin
      A[i,j]:=j-2.0;
      A[j,i]:=j-2.0;
    end;
    A[i,i]:=i;
    if m=n then sym:=true;
  end;
end;
4: begin {Frank symmetric}
  for i:=1 to mn do
    for j:=1 to i do
    begin
      A[i,j]:=j;
      A[j,i]:=j;
    end;
    if m=n then sym:=true;
end;
5: begin {Bordered}
  temp:=2.0;
```

```
    for i:=1 to (mn-1) do
    begin
      temp:=temp/2.0; {2^(1-i)}
      for j:=1 to mn do
        A[i,j]:=0.0;
      A[i,mn]:=temp;
      A[mn,i]:=temp;
      A[i,i]:=1.0;
    end;
    A[mn,mn]:=1.0;
    if m=n then sym:=true;
  end;
  6: begin {Diagonal}
    for i:=1 to mn do
    begin
      for j:=1 to mn do
        A[i,j]:=0.0;
      A[i,i]:=i;
    end;
    if m=n then sym:=true;
  end;
  7: begin {W+}
    k:=mn div 2; {[n/2]}
    for i:=1 to mn do
      for j:=1 to mn do
        A[i,j]:=0.0;
    if m=n then sym:=true;
    for i:=1 to k do
    begin
      A[i,i]:=k+1-i;
      A[mn-i+1,mn-i+1]:=k+1-i;
    end;
    for i:=1 to mn-1 do
    begin
      A[i,i+1]:=1.0;
      A[i+1,i]:=1.0;
    end;
  end;
  8: begin {W-}
    k:=mn div 2; {[n/2]}
    for i:=1 to mn do
      for j:=1 to mn do
        A[i,j]:=0.0;
    if m=n then sym:=true;
    for i:=1 to k do
    begin
      A[i,i]:=k+1-i;
      A[mn-i+1,mn-i+1]:=i-1-k;
    end;
    for i:=1 to mn-1 do
    begin
      A[i,i+1]:=1.0;
      A[i+1,i]:=1.0;
```

```
      end;
      if m=n then sym:=true;
    end;
    9: begin {Constant}
      write('Set all elements to a constant value = ');
      readln(temp);
      writeln(temp);
      for i:=1 to mn do
        for j:=1 to mn do
          A[i,j]:=temp;
      if m=n then sym:=true;
    end;
    10: begin {Unit}
      for i:=1 to mn do
      begin
        for j:=1 to mn do A[i,j]:=0.0;
        A[i,i]:=1.0;
      end;
      if m=n then sym:=true;
    end;
    else {case statement else} {!!!! Note missing close bracket here}
    begin
      writeln;
      writeln('*** ERROR *** unrecognized option');
      halt;
    end; {else of case statement}
  end; {case statement}
  if sym then
  begin {convert to vector form}
    k:=0; {index for vector element}
    for i:=1 to n do
    begin
      for j:=1 to i do
      begin
        k:=k+1;
        avector[k]:=A[i,j];
      end;
    end;
  end;
end; {matrixin}

procedure vectorin(n: integer; var Y: rvector);
{vectorin.pas
  == enter or generate a vector of n real elements
}
var
  i, j, k, m, nt : integer;
  x : real;

begin
  write('vectorin.pas');
  writeln('   -- enter or generate a real vector of ',n,' elements');
  writeln('Options:');
```

```
  writeln('   1) constant');
  writeln('   2) uniform random in [0,user_value) ');
  writeln('   3) user entered from console ');
  writeln('   4) entered from RHS columns in matrix file ');
  write('   Choose option :');
  readln(i);
  writeln(i);
  Case i of
    1 : begin
        write('Enter constant value =');
        readln(x);
        writeln(x);
        for j:=1 to n do Y[j]:=x;
    end;
    2 : begin
        write('Enter the upper bound to the generator =');
        readln(x);
        writeln(x);
        for j:=1 to n do Y[j]:=Random;
        {According to the Turbo Pascal manual, version 3.0, Random
          returns a number in [0,1). My experience is that most such
          pseudo-random number generators leave a lot to be desired
          in terms of statistical properties. I do NOT recommend it
          for serious use in Monte Carlo calculations or other situations
          where a quality generator is required. For a better generator
          in Pascal, see Wichman B. and Hill, D., (1987)}
    end;
    3 : begin
        writeln('Enter elements of vector one by one');
        for j:=1 to n do
        begin
          write('Y[',j,']=');
          readln(Y[j]);
          writeln(Y[j]);
        end;
    end;
  end {case};
end {vectorin.pas};

function resids(nRow, nCol: integer; A : rmatrix;
          Y: rvector; Bvec : rvector; print : boolean):real;
{resids.pas
  == Computes residuals and , if print is TRUE, displays them 7
    per line for the linear least squares problem. The sum of
    squared residuals is returned.

    residual vector = A * Bvec - Y
}
var
i, j: integer;
t1, ss : real;

begin
```

```
  if print then
  begin
    writeln('Residuals');
  end;
  ss:=0.0;
  for i:=1 to nRow do
  begin
    t1:=-Y[i]; {note form of residual is residual = A * B - Y }
    for j:=1 to nCol do
      t1:=t1+A[i,j]*Bvec[j];
    ss:=ss+t1*t1;
    if print then
    begin
      write(t1:10,' ');
      if (i = 7 * (i div 7)) and (i<nRow) then writeln;
    end;
  end; {loop on i}
  if print then
  begin
    writeln;
    writeln('Sum of squared residuals =',ss);
  end;
  resids:=ss
end; {resids.pas == residual calculation for linear least squares}

procedure choldcmp(n: integer;
                   var a: smatvec;
                   var singmat: boolean);



var
  i,j,k,m,q: integer;
  s : real;

begin
  singmat := false;
  for j := 1 to n do
  begin
    q := j*(j+1) div 2;
    if j>1 then
    begin
      for i := j to n do
      begin
        m := (i*(i-1) div 2)+j; s := a[m];
        for k := 1 to (j-1) do s := s-a[m-k]*a[q-k];
        a[m] := s;
      end;
    end;
    if a[q]<=0.0 then
    begin
      singmat := true;
      a[q] := 0.0;
    end;
```

```
    s := sqrt(a[q]);
    for i := j to n do
    begin
      m := (i*(i-1) div 2)+j;
      if s=0.0 then a[m] := 0
          else a[m] := a[m]/s;
    end;
  end;
end;

procedure cholback(n: integer;
                   a: smatvec;
                   var x: rvector);


var
 i,j,q  : integer;

begin
  if a[1]=0.0 then x[1]:=0.0
              else x[1]:=x[1]/a[1];
  if n>1 then
  begin
    q:=1;
    for i:=2 to n do
    begin
      for j:=1 to (i-1) do
      begin
        q:=q+1; x[i]:=x[i]-a[q]*x[j];
      end;
      q:=q+1;
      if a[q]=0.0 then x[i]:=0.0
                  else x[i]:=x[i]/a[q];
    end;
  end;

  if a[n*(n+1) div 2]=0.0 then x[n]:=0.0
                          else x[n]:=x[n]/a[n*(n+1) div 2];
  if n>1 then
  begin
    for i:=n downto 2 do
    begin
      q:=i*(i-1) div 2;
      for j:=1 to (i-1) do x[j]:=x[j]-x[i]*a[q+j];
      if a[q]=0.0 then x[i-1]:=0.0
                  else x[i-1]:=x[i-1]/a[q];
    end;
  end;
end;

var
  A : rmatrix;
  avector : smatvec;
```

```
  i, j, k, nCol, nRow : integer;
  sym : boolean;
  Y, Ycopy : rvector; {to store the right hand side of the equations}
  singmat : boolean; {set true if matrix discovered to be computationally
              singular during alg07.pas}
  s : real; {an accumulator}

begin
  banner:='dr0708 -- Choleski decomposition and back-substitution';
  write('order of problem = ');
  readln(nRow);
  writeln(nRow);
  nCol:=nRow; {use symmetric matrix in Choleski}
  Matrixin(nRow,nCol,A,avector,sym);
  writeln;
  writeln('returned matrix of order ',nRow);
  if not sym then halt; {must have symmetric matrix}
  begin
    writeln('Symmetric matrix -- Vector form');
    k:=0;
    for i:=1 to nRow do
    begin
      for j:=1 to i do
      begin
         k:=k+1;
        write(avector[k]:10:5,' ');
        if (7 * (j div 7) = j) and (j<i) then writeln;
      end;
      writeln;
    end;
  end;
  writeln('Enter right hand side of equations');
  vectorin(nRow, Y);
  for i:=1 to nRow do Ycopy[i]:=Y[i];
  writeln;
  choldcmp(nRow,avector, singmat); {decompose matrix}
  begin
    writeln('Decomposed matrix -- Vector form');
    k:=0;
    for i:=1 to nRow do
    begin
      for j:=1 to i do
      begin
        k:=k+1;
        write(avector[k]:10:5,' ');
        if (7 * (j div 7) = j) and (j<i) then writeln;
      end;
      writeln;
    end;
  end;
  if not singmat then
  begin
    Cholback(nRow,avector,Y);
```

```
    writeln('Solution');
    for i:=1 to nRow do
    begin
      write(Y[i]:10:5,' ');
      if (7 * (i div 7) = i) and (i<nRow) then writeln;
      writeln;
    end;
    s:=resids(nRow,nCol,A,Ycopy,Y,true);
  end {non-singular case}
  else
  begin
    writeln('Matrix computationally singular -- solution not possible');
  end;
end. {dr0708.pas}
```

Example output

```
fpc ../Pascal2021/dr0708.pas
# copy to run file
mv ../Pascal2021/dr0708 ../Pascal2021/dr0708.run
../Pascal2021/dr0708.run <../Pascal2021/dr0708p.in >../Pascal2021/dr0708p.out
```

```
## Free Pascal Compiler version 3.0.4+dfsg-23 [2019/11/25] for x86_64
## Copyright (c) 1993-2017 by Florian Klaempfl and others
## Target OS: Linux for x86-64
## Compiling ../Pascal2021/dr0708.pas
## dr0708.pas(290,9) Note: Local variable "k" not used
## dr0708.pas(290,12) Note: Local variable "m" not used
## dr0708.pas(290,15) Note: Local variable "nt" not used
## dr0708.pas(461,3) Note: Local variable "s" is assigned but never used
## Linking ../Pascal2021/dr0708
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 522 lines compiled, 0.1 sec
## 4 note(s) issued
```

```
order of problem = 5
Matrixin.pas -- generate or input a real matrix 5 by 5
Possible matrices to generate:
0) Keyboard or console file input
1) Hilbert segment
2) Ding Dong
3) Moler
4) Frank symmetric
5) Bordered symmetric
6) Diagonal
7) Wilkinson W+
8) Wilkinson W-
9) Constant
10) Unit
Enter type to generate 3

returned matrix of order 5
Symmetric matrix -- Vector form
   1.00000
```

```
  -1.00000    2.00000
  -1.00000    0.00000    3.00000
  -1.00000    0.00000    1.00000    4.00000
  -1.00000    0.00000    1.00000    2.00000    5.00000
Enter right hand side of equations
vectorin.pas   -- enter or generate a real vector of 5 elements
Options:
   1) constant
   2) uniform random in [0,user_value)
   3) user entered from console
   4) entered from RHS columns in matrix file
   Choose option :1
Enter constant value = 1.0000000000000000E+000

Decomposed matrix -- Vector form
   1.00000
  -1.00000    1.00000
  -1.00000   -1.00000    1.00000
  -1.00000   -1.00000   -1.00000    1.00000
  -1.00000   -1.00000   -1.00000   -1.00000    1.00000
Solution
 171.00000
  86.00000
  44.00000
  24.00000
  16.00000
Residuals
 0.00E+000  0.00E+000  0.00E+000  0.00E+000  0.00E+000
Sum of squared residuals = 0.0000000000000000E+000
```

Algorithm 9 – Bauer-Reinsch matrix inversion

Wilkinson, Reinsch, and Bauer (1971), pages 45-49, is a contribution entitled **Inversion of Positive Definite Matrices by the Gauss-Jordan Method**. It hardly mentions, but appears to assume, that the matrix to be inverted is symmetric. Two Algol procedures are provided, one for a matrix stored as a square array, the other for the a matrix where only the lower triangle is stored as a single vector in row-wise order. That is, if A is of order n=3 and has values

```
1  2  4
2  3  5
4  5  6
```

Then the corresponding vector of 6 = n*(n+1)/2 values is

```
 1  2  3  4  5  6
```

By some exceedingly clever coding and matrix manipulation, Bauer and Reinsch developed tiny codes that invert a positive-definite matrix *in situ* using only one extra vector of length n. Thus, besides the memory to store a very small code, we need only n*(n+3)/2 floating point numbers and a few integers to index arrays.

Truthfully, we rarely need an explicit matrix inverse, and the most common positive-definite symmetric matrix that arises in scientific computations is the sum of squares and cross-products (SSCP) in the normal equations used for linear (or also nonlinear) least squares problems. However, the formation of this SSCP matrix is rarely the best approach to solving least squares problems. The SVD introduced in Algorithm 1 and the least squares solution in Algorithm 2 lead to better methods. (??mention A4, Choleski in A7, A8 etc.)

Despite these caveats, the Bauer-Reinsch algorithm is interesting as a historical curiosity, showing what can be done when resources are very limited.

Fortran

Listing

```
C&&& A9
C  TEST ALGORITHM 9  A9GJ
C  J.C. NASH   JULY 1978, APRIL 1989
C  USE FRANK MATRIX
      LOGICAL INDEF
      INTEGER N,N2,I,J,IJ,NOUT
      REAL A(55),X(10),S,T
      N2=55
C  PRINTER CHANNEL
      NOUT=6
C  MAIN LOOP
C       DO 100 N=2,10,2
      N = 4
      WRITE(NOUT,950)N
 950  FORMAT('0ORDER=',I4,'  ORIGINAL MATRIX')
C  PUT IN CARDS FROM A78
C     NOTE DIFFERENCES ONLY IN CALLS
        DO 20 I=1,N
          DO 10 J=1,I
            IJ=I*(I-1)/2+J
            A(IJ)=J
  10      CONTINUE
  20    CONTINUE
       CALL SOUT(A,N2,N,NOUT)
```

```
      CALL A9GJ(A,N2,N,INDEF,X)
      WRITE(NOUT,956)
 956  FORMAT('0INVERSE')
      CALL SOUT(A,N2,N,NOUT)
      WRITE(NOUT,957)
 957  FORMAT('0INVERSE OF INVERSE')
      CALL A9GJ(A,N2,N,INDEF,X)
      CALL SOUT(A,N2,N,NOUT)
C   COMPUTE DEVIATION FROM ORIGINAL MATRIX
       S=0.0
       DO 50 I=1,N
         DO 40 J=1,I
         IJ=I*(I-1)/2+J
         T=ABS(J-A(IJ))
         IF(T.GT.S)S=T
  40    CONTINUE
  50  CONTINUE
      WRITE(NOUT,958)S
 958  FORMAT('0MAX. DEVN. OF INVERSE-INVERSE FROM ORIGINAL=',1PE16.8)
C 100  CONTINUE
     STOP
     END
     SUBROUTINE SOUT(A,N2,N,NOUT)
C  J.C. NASH   JULY 1978, APRIL 1989
     INTEGER N2,N,NOUT,I,J,IJ,JJ
     REAL A(N2)
C     PRINTS SYMMETRIC MATRIX STORED ROW-WISE AS A VECTOR
     DO 20 I=1,N
       WRITE(NOUT,951)I
 951   FORMAT(' ROW',I3)
       IJ=I*(I-1)/2+1
       JJ=IJ+I-1
       WRITE(NOUT,952)(A(J),J=IJ,JJ)
 952   FORMAT(1H ,1P5E16.8)
 20  CONTINUE
     RETURN
     END
     SUBROUTINE A9GJ(A,N2,N,INDEF,X)
C  ALGORITHM 9
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  BAUER-REINSCH  GAUSS-JORDAN INVERSION OF A SYMMETRIC, POSITIVE
C  A=MATRIX - STORED AS A VECTOR -- ELEMENT I,J IN POSITION I*(I-1)/2+J
C  N2=LENGTH OF VECTOR A = N*(N+1)/2
C  N=ORDER OF MATRIX
C  INDEF=LOGICAL FLAG SET .TRUE. IF MATRIX NOT COMPUTATIONALLY
C     POSITIVE DEFINITE
C  X=WORKING VECTOR OF LENGTH AT LEAST N
C  DEFINITE MATRIX
C  STEP 0
     LOGICAL INDEF
     INTEGER N2,N,K,KK,Q,M,Q2,JI,JQ
     REAL A(N2),S,T,X(N)
C  STEP 1
```

```
      INDEF=.FALSE.
      DO 100 KK=1,N
        K=N+1-KK
C   STEP 2
        S=A(1)
C   STEP 3
        IF(S.LE.0.0)INDEF=.TRUE.
        IF(INDEF)RETURN
C   STEP 4
        M=1
C   STEP 5
        DO 60 I=2,N
C   STEP 6
          Q=M
          M=M+I
          T=A(Q+1)
          X(I)=-T/S
C   STEP 7
          Q2=Q+2
          IF(I.GT.K)X(I)=-X(I)
C   STEP 8
          DO 40 J=Q2,M
            JI=J-I
            JQ=J-Q
            A(JI)=A(J)+T*X(JQ)
  40      CONTINUE
C   STEP 9
  60    CONTINUE
C   STEP 10
        Q=Q-1
        A(M)=1/S
C   STEP 11
        DO 80 I=2,N
          JI=Q+I
          A(JI)=X(I)
  80    CONTINUE
C   STEP 12
 100  CONTINUE
      RETURN
      END
```

Example output

```
## #!/bin/bash
gfortran ../fortran/a9.f
mv ./a.out ../fortran/a9.run
../fortran/a9.run
```

```
## OORDER=   4  ORIGINAL MATRIX
##  ROW  1
##    1.00000000E+00
##  ROW  2
##    1.00000000E+00  2.00000000E+00
##  ROW  3
```

```
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW  4
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00
## OINVERSE
##  ROW  1
##    2.00000000E+00
##  ROW  2
##   -1.00000000E+00  2.00000000E+00
##  ROW  3
##    0.00000000E+00 -1.00000000E+00  2.00000000E+00
##  ROW  4
##    0.00000000E+00  0.00000000E+00 -1.00000000E+00  1.00000000E+00
## OINVERSE OF INVERSE
##  ROW  1
##    1.00000012E+00
##  ROW  2
##    1.00000024E+00  2.00000048E+00
##  ROW  3
##    1.00000036E+00  2.00000072E+00  3.00000095E+00
##  ROW  4
##    1.00000036E+00  2.00000072E+00  3.00000095E+00  4.00000095E+00
## OMAX. DEVN. OF INVERSE-INVERSE FROM ORIGINAL=  9.53674316E-07
```

BASIC

Listing

```
10 PRINT "ALGORITHM 9 - BAUER REINSCH INVERSION TEST"
20 N=100
40 DIM A(N*(N+1)/2),X(N)
45 LET N=4
50 GOSUB 1500
51 REM BUILD MATRIX IN A
60 GOSUB 1400
61 REM PRINT IT
70 GOSUB 1000
71 REM INVERT
80 GOSUB 1400
81  REM PRINT
90 quit
110 STOP
1000 REM ALG. 9 BAUER REINSCH INVERSION
1010 FOR K=N TO 1 STEP -1
1011     REM STEP 1
1020   S=A(1)
1021     REM STEP 2
1030   IF S<=0 THEN EXIT 1160
1031     REM STEP 3
1040   M=1
1041     REM STEP 4
1050   FOR I=2 TO N
1051     REM STEP 5
1060      Q=M
1061     M=M+I
```

```
1062     T=A(Q+1)
1063     X(I)=-T/S
1064     REM STEP 6
1070      IF I>K THEN X(I)=-X(I)
1071     REM STEP 7
1080      FOR J=Q+2 TO M
1081     REM STEP 8
1090         A(J-I)=A(J)+T*X(J-Q)
1100      NEXT J
1110   NEXT I
1111     REM STEP 9
1120   Q=Q-1
1121     A(M)=1/S
1122     REM STEP 10
1130   FOR I=2 TO N
1131     A(Q+I)=X(I)
1132     NEXT I
1133     REM STEP 11
1140 NEXT K
1141     REM STEP 12
1150 RETURN
1160 PRINT "MATRIX COMPUTATIONALLY INDEFINITE"
1170 STOP
1171     REM END ALG. 9
1400 PRINT "MATRIX A"
1410 FOR I=1 TO N
1420 FOR J=1 TO I
1430 PRINT A(I*(I-1)/2+J);
1440 NEXT J
1450 PRINT
1460 NEXT I
1470 RETURN
1500 REM FRANK MATRIX
1510 FOR I=1 TO N
1520 FOR J=1 TO I
1530 LET A(I*(I-1)/2+J)=J
1540 NEXT J
1550 NEXT I
1560 RETURN
```

Example output

```
bwbasic ../BASIC/a9.bas >../BASIC/a9.out
# echo "done"
```

```
Bywater BASIC Interpreter/Shell, version 2.20 patch level 2

Copyright (c) 1993, Ted A. Campbell

Copyright (c) 1995-1997, Jon B. Volkoff


ALGORITHM 9 - BAUER REINSCH INVERSION TEST
```

```
MATRIX A
 1
 1 2
 1 2 3
 1 2 3 4
MATRIX A
 2
 -1 2
 0 -1 2
 0 0 -1 1
```

Pascal

Listing

```
program dr09(input,output);
{dr09.pas == driver program to test procedure for the Bauer-Reinsch
          inversion of a symmetric positive definite real matrix stored
          in row-wise vector form

          Copyright 1988 J.C.Nash
}
{I constype.def}
{constype.def ==
  This file contains various definitions and type statements which are
  used throughout the collection of "Compact Numerical Methods".  In many
  cases not all definitions are needed, and users with very tight memory
  constraints may wish to remove some of the lines of this file when
  compiling certain programs.

  Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}
{uses Dos, Crt;} {Turbo Pascal 5.0 Modules}
{ 1. Interrupt, Unit, Interface, Implementation, Uses are reserved words now.}
{ 2. System,Dos,Crt are standard unit names in Turbo 5.0.}

const
  big = 1.0E+35;    {a very large number}
  Maxconst = 25;    {Maximum number of constants in data record}
  Maxobs = 100;     {Maximum number of observations in data record}
  Maxparm  = 25;    {Maximum number of parameters to adjust}
  Maxvars = 10;     {Maximum number of variables in data record}
  acctol = 0.0001;  {acceptable point tolerance for minimisation codes}
  maxm = 20;        {Maximum number or rows in a matrix}
  maxn = 20;        {Maximum number of columns in a matrix}
  maxmn = 40;       {maxn+maxm, the number of rows in a working array}
  maxsym = 210;     {maximum number of elements of a symmetric matrix
              which need to be stored = maxm * (maxm + 1)/2 }
  reltest = 10.0;   {a relative size used to check equality of numbers.
              Numbers x and y are considered equal if the
              floating-point representation of reltest+x equals
```

```
            that of reltest+y.}
  stepredn = 0.2;   {factor to reduce stepsize in line search}
  yearwrit = 1990;  {year in which file was written}

type
  str2 = string[2];
  rmatrix = array[1..maxm, 1..maxn] of real; {a real matrix}
  wmatrix = array[1..maxmn, 1..maxn] of real; {a working array, formed
                as one real matrix stacked on another}
  smatvec = array[1..maxsym] of real; {a vector to store a symmetric matrix
            as the row-wise expansion of its lower triangle}
  rvector = array[1..maxm] of real;  {a real vector. We will use vectors
            of m elements always. While this is NOT space efficient,
            it simplifies program codes.}
  cgmethodtype= (Fletcher_Reeves,Polak_Ribiere,Beale_Sorenson);
    {three possible forms of the conjugate gradients updating formulae}
  probdata = record
          m     : integer; {number of observations}
          nvar  : integer; {number of variables}
          nconst: integer; {number of constants}
          vconst: array[1..Maxconst] of real;
          Ydata : array[1..Maxobs, 1..Maxvars] of real;
          nlls  : boolean; {true if problem is nonlinear least squares}
        end;
{
  NOTE: Pascal does not let us define the work-space for the function
  within the user-defined code.  This is a weakness of Pascal for this
  type of work.
}
var {global definitions}
  banner     : string[80]; {program name and description}

Procedure Frank(var n: integer; var A: rmatrix; var avector: smatvec);
var
  i,j: integer;
begin
  writeln('Frank symmetric');
    for i:=1 to n do
    begin
        for j:=1 to i do
        begin
          A[i,j]:=j;
          A[j,i]:=j;
        end;
    end;
end;

Procedure mat2vec(var n: integer; var A: rmatrix; var avector: smatvec);
var
  i,j,k: integer;

begin {convert to vector form}
    k:=0; {index for vector element}
```

```
    for i:=1 to n do
    begin
      for j:=1 to i do
      begin
        k:=k+1;
        avector[k]:=A[i,j];
      end;
    end;
end; {matrixin}

Procedure vec2mat(var n: integer; var A: rmatrix; var avector: smatvec);
var
  i,j,k: integer;

  begin {convert to matrix form}
    k:=0; {index for vector element}
    for i:=1 to n do
    begin
      for j:=1 to i do
      begin
        k:=k+1;
        A[i,j]:=avector[k];
      end;
    end;
end; {matrixin}

{ I alg09.pas}
procedure brspdmi(n : integer;
                var avector : smatvec;
                var singmat : boolean);

var
  i,j,k,m,q : integer;
  s,t : real;
  X : rvector;

begin
  writeln('alg09.pas -- Bauer Reinsch inversion');
  singmat  :=  false;
  for k  :=  n downto 1 do
  begin
    if (not singmat) then
    begin
      s  :=  avector[1];
      if s>0.0 then
      begin
        m  :=  1;
        for i := 2 to n do
        begin
          q := m; m := m+i; t := avector[q+1]; X[i] := -t/s;

          if i>k then X[i] := -X[i];
          for j := (q+2) to m do
```

```
          begin
            avector[j-i] := avector[j]+t*X[j-q];
          end;
        end;
        q := q-1; avector[m] := 1.0/s;
        for i := 2 to n do avector[q+i] := X[i];
      end
      else
        singmat := true;
    end;
  end;
end;

var
  A, Ainverse : rmatrix;
  avector : smatvec;
  i, imax, j, jmax, k, n : integer;
  errmax, s : real;
  singmat: boolean;


BEGIN { main program }
  banner:='dr09.pas -- test Bauer Reinsch sym, posdef matrix inversion';
  writeln(banner);
  n:=4; {Fixed example size 20210113}
  Frank(n,A,avector);
  writeln;
  writeln('returned matrix of order ',n);
  begin
    for i:=1 to n do
    begin
        for j:=1 to n do
        begin
            write(A[i,j],' ');
        end;
        writeln;
    end;
  end;
  mat2vec(n, A, avector);
  begin
    writeln('Symmetric matrix -- Vector form');
    k := 0;
    for i := 1 to n do
    begin
      for j := 1 to i do
      begin
        k := k+1;
        write(avector[k]:10:5,' ');
      end;
      writeln;
    end;
  end;
  brspdmi(n, avector,singmat);
```

```
  if singmat then halt; {safety check}
  writeln('Computed inverse');
  k := 0; {initialize index to smatvec elements}
  for i := 1 to n do
  begin
    for j := 1 to i do
    begin
      k := k+1;
      write(avector[k]:10:5,' ');
      Ainverse[i,j] := avector[k]; {save square form of inverse}
      Ainverse[j,i] := avector[k];
      if (7 * (j div 7) = j) and (j<i) then
      begin
        writeln;
      end;
    end;
    writeln;
  end;
  {Compute maximum error in A * Ainverse and note where it occurs.}
  errmax := 0.0; imax := 0; jmax := 0;
  for i := 1 to n do
  begin
    for j := 1 to n do
    begin
      s := 0.0; if i=j then s := -1.0;
      for k := 1 to n do s := s + Ainverse[i,k]*A[k,j];
      {Note: A has not been altered, since avector was used.}
      if abs(s)>abs(errmax) then
      begin
        errmax := s; imax := i; jmax := j; {save maximum error, indices}
      end;
    end; {loop on j}
  end; {loop on i}
  writeln('Maximum element in Ainverse * A - 1(n) = ',errmax,
          '  position ',imax,',',jmax);
end. {dr09.pas == Bauer Reinsch inversion}
```

Example output

```
fpc ../Pascal2021/dr09.pas
# copy to run file
mv ../Pascal2021/dr09 ../Pascal2021/dr09.run
../Pascal2021/dr09.run >../Pascal2021/dr09p.out
```

```
## Free Pascal Compiler version 3.0.4+dfsg-23 [2019/11/25] for x86_64
## Copyright (c) 1993-2017 by Florian Klaempfl and others
## Target OS: Linux for x86-64
## Compiling ../Pascal2021/dr09.pas
## Linking ../Pascal2021/dr09
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 233 lines compiled, 0.1 sec
```

```
dr09.pas -- test Bauer Reinsch sym, posdef matrix inversion
Frank symmetric
```

```
returned matrix of order 4
 1.0000000000000000E+000  1.0000000000000000E+000  1.0000000000000000E+000  1.0000000000000000E+000
 1.0000000000000000E+000  2.0000000000000000E+000  2.0000000000000000E+000  2.0000000000000000E+000
 1.0000000000000000E+000  2.0000000000000000E+000  3.0000000000000000E+000  3.0000000000000000E+000
 1.0000000000000000E+000  2.0000000000000000E+000  3.0000000000000000E+000  4.0000000000000000E+000
Symmetric matrix -- Vector form
   1.00000
   1.00000    2.00000
   1.00000    2.00000    3.00000
   1.00000    2.00000    3.00000    4.00000
alg09.pas -- Bauer Reinsch inversion
Computed inverse
   2.00000
  -1.00000    2.00000
   0.00000   -1.00000    2.00000
   0.00000    0.00000   -1.00000    1.00000
Maximum element in Ainverse * A - 1(n) =  0.0000000000000000E+000  position 0,0
```

Python

WARNING: interim test only!!!???

Listing

The Algorithm 9 code:

```
# -*- coding: utf-8 -*-
"""
CNM Algorithm 09 test

J C Nash 2021-1-12
"""

import numpy
import math
import sys
def brspdmi(Avec, n):
# ================================================================================
# Bauer Reinsch inverse of symmetric positive definite matrix stored
#    as a vector that has the lower triangle of the matrix in row order
# ================================================================================
    print(Avec)
    X = numpy.array([ 0 ] * n) # zero vector x
    for k in range(n, 0, -1):
        s  =  Avec[0];
        #print("s=",s)
        if (s > 0.0) :
            m  =  1;
            for i in range(2,n+1):
                q = m
                m = m+i
                t = Avec[q]
                X[i-1] = -t/s
                if i>k :
```

```
                    X[i-1] = -X[i-1]
            #     print("i, q, m:", i, q, m)
                for j in range(q+2), m+1):
            #         print(j)
            #         print("j-q-1=",j-q-1)
            #         print(X[j-q-1])
                    Avec[j-i-1] = Avec[j-1]+t*X[j-q-1]
                q = q-1
                Avec[m-1] = 1.0/s
            for i in range(2, n+1):
                print("i ",i)
                Avec[q+i-1] = X[i-1]
        else :
            print("Matrix is singular")
            sys.exit()
        print(k,":",Avec)
    return(Avec)

def FrankMat(n):
    Amat = numpy.array([ [ 0 ] * n ] * n) # numpy.empty(shape=(n,n), dtype='object')
    for i in range(1,n+1):
#        print("i=",i)
        for j in range(1,i+1):
#            print(j)
            Amat[i-1,j-1]=j
            Amat[j-1,i-1]=j
    return(Amat)

def smat2vec(Amat):
    n=len(Amat[0])
    n2=int(n*(n+1)/2)
    svec = [ None ] * n2
    k = 0
    for i in range(1,n+1):
        for j in range(1,i+1):
            svec[k]=Amat[i-1, j-1]
            k=k+1
    return(svec)

def svec2mat(svec):
    n2=len(svec)
    n=int((-1+math.sqrt(1+8*n2))/2)
    print("matrix is of size ",n)
    Amat = numpy.array([ [ None ] * n ] * n)
    k = 0
    for i in range(1,n+1):
        for j in range(1,i+1):
            Amat[i-1, j-1] = svec[k]
            Amat[j-1, i-1] = svec[k]
            k=k+1
    return(Amat)

# Main program
```

```
AA = FrankMat(4)
print(AA)
avec = smat2vec(AA)
print(avec)
n=len(AA[0])
vinv = brspdmi(avec, n)
## Computed inverse
##    2.00000
##   -1.00000    2.00000
##    0.00000   -1.00000    2.00000
##    0.00000    0.00000   -1.00000    1.00000



print(vinv)
Ainv = svec2mat(vinv)
print(Ainv)
print(AA)
print(numpy.dot(Ainv, AA))
```

Example output

```
python3 ../python/A9.py
```

```
## [[1 1 1 1]
##  [1 2 2 2]
##  [1 2 3 3]
##  [1 2 3 4]]
## [1, 1, 2, 1, 2, 3, 1, 2, 3, 4]
## [1, 1, 2, 1, 2, 3, 1, 2, 3, 4]
## i  2
## i  3
## i  4
## 4 : [1, 1, 2, 1, 2, 3, -1, -1, -1, 1.0]
## i  2
## i  3
## i  4
## 3 : [1, 1, 2, 0, 0, 2.0, -1, -1, -1, 1.0]
## i  2
## i  3
## i  4
## 2 : [1, 0, 2.0, 0, -1, 2.0, -1, 0, -1, 1.0]
## i  2
## i  3
## i  4
## 1 : [2.0, -1, 2.0, 0, -1, 2.0, 0, 0, -1, 1.0]
## [2.0, -1, 2.0, 0, -1, 2.0, 0, 0, -1, 1.0]
## matrix is of size  4
## [[2.0 -1 0 0]
##  [-1 2.0 -1 0]
##  [0 -1 2.0 -1]
##  [0 0 -1 1.0]]
## [[1 1 1 1]
```

```
##  [1 2 2 2]
##  [1 2 3 3]
##  [1 2 3 4]]
## [[1.0 0.0 0.0 0.0]
##  [0.0 1.0 0.0 0.0]
##  [0.0 0.0 1.0 0.0]
##  [0.0 0.0 0.0 1.0]]
```

R

Listing and Example output

```
A9 <- function(a, n){
    x <- rep(0, n)
    for (k in n:1){
        s=a[1]
        if (s <= 0){
          stop("A9: matrix is singular")
        }
       m<-1
       for (i in 2:n){
         q<-m; m<-m+i; t<-a[q+1]; x[i]<--t/s
         if (i > k) { x[i] <- -x[i]}
         for (j in (q+2):m){
           a[j-i]<-a[j]+t*x[j-q]
         }
       }
     q<-q-1; a[m]=1/s
     for (i in 2:n){a[q+i] <- x[i]}
#    cat("iteration k:")
#     print(a)
    }
    a
}

FrankMat <- function(n){
  Amat <- matrix(0, nrow=n, ncol=n)
  for (i in 1:n){
     for (j in 1:i){
         Amat[i,j]<-j; Amat[j,i]<-j
     }
  }
    Amat
}

smat2vec <- function(Amat){
   n<-dim(Amat)[1]
   n2<-(n*(n+1)/2)
   svec = rep(0, n2)
   k <- 0
  for (i in 1:n){
    for (j in 1:i){
       k<-k+1
       svec[k]<-Amat[i,j]
```

```
    }
  }
  svec
}

svec2mat <- function(svec){
  n2<-length(svec)
  n <- (-1+sqrt(1+8*n2))/2
  Amat <- matrix(0, nrow=n, ncol=n)
  k <- 0
  for (i in 1:n){
    for (j in 1:i){
      k<-k+1
      Amat[j,i]<-Amat[i,j]<-svec[k]
    }
  }
  Amat
}

n <- 4
AA <- FrankMat(n)
vv <- smat2vec(AA)
vv
```

```
##  [1] 1 1 2 1 2 3 1 2 3 4
```

```
vinv<-A9(vv, n)
vinv
```

```
##  [1]  2 -1  2  0 -1  2  0  0 -1  1
```

```
print(vinv)
```

```
##  [1]  2 -1  2  0 -1  2  0  0 -1  1
```

```
Ainv<-svec2mat(vinv)
print(Ainv)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2   -1    0    0
## [2,]   -1    2   -1    0
## [3,]    0   -1    2   -1
## [4,]    0    0   -1    1
```

```
print(Ainv %*% AA)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

Others

Algorithm 10 – Inverse iteration via Gaussian elimination

The purpose of this algorithm is to find a single eigensolution of a matrix A via inverse iteration. That is, we want solutions (e,x) of

$$Ax = ex$$

The programs do not require a symmetric matrix, which leaves open the possibility that a solution may not exist in the unsymmetric case.

Fortran

The Algorithm 10 code:

```
C  TEST ALGORITHM 10 USING HILBERT SEGMENT
C  J.C. NASH   JULY 1978, APRIL 1989
      INTEGER N,N2,I,J,JN,NOUT
      REAL W(10,20),X(10),Y(10),SHIFT,EPS,S
C   PRINTER CHANNEL
        NOUT=6
C   MACHINE PRECISION      NOTE -- IBM SHORT PRECISION
      EPS=16.0**(-5)
C   MAIN LOOP
      DO 100 N=5,10,5
C   CREATE HILBERT SEGMENT & UNIT MATRIX
        N2=2*N
        DO 20 I=1,N
          DO 10 J=1,N
            JN=J+N
            W(I,J)=1.0/(I+J-1)
            W(I,JN)=0.0
  10      CONTINUE
          JN=N+I
          W(I,JN)=1.0
  20    CONTINUE
        WRITE(NOUT,950)N
 950    FORMAT(' ORDER=',I4)
        SHIFT=0.0
        WRITE(NOUT, 960)SHIFT
 960    FORMAT(' USING SHIFT OF ',F12.6)
C   SET GUESS TO VECTOR
        DO 30 I=1,N
           X(I)=1.0
  30   CONTINUE
C   SHOULD INPUT SHIFT
        LIMIT=10*N
C   SET OUTPUT CHANNEL TO 0 TO SUPPRESS OUTPUT (ELSE USE NOUT)
        CALL A10GII(W,10,N,N2,X,Y,SHIFT,EPS,LIMIT,EV,0)
        WRITE(NOUT,951)EV,LIMIT
 951    FORMAT(' CONVERGED TO EV=',1PE16.8,'  IN ',I4,'  ITERATIONS')
        WRITE(NOUT,952)(I,X(I),I=1,N)
 952    FORMAT(' X(',I3,')=',E16.8)
        DO 50 I=1,N
          S=EV*X(I)
          DO 40 J=1,N
```

```
          S=S-X(J)/(I+J-1)
 40     CONTINUE
        WRITE(NOUT,953)I,S
953     FORMAT(' RESIDUAL(',I3,')=',E16.8)
 50   CONTINUE
100 CONTINUE
    STOP
    END
    SUBROUTINE A10GII(W,NW,N,N2,X,Y,SHIFT,EPS,LIMIT,EV,IPR)
C  ALGORITHM 10
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  INVERSE ITERATION VIA GAUSS ELIMINATION
C  SOLVES EIGENPROBLEM A*X=EV*B*X FOR EIGENSOLUTION (EX,X)
C  VECTOR NORMALISED SO LARGEST ELEMENT IS 1.0
C  W=WORKING ARRAY HAVING INITIALLY A IN COLUMNS 1 TO N
C                                   B IN COLUMNS N+1 TO 2*N=N2
C  NW=FIRST DIMENSION OF W
C  N=ORDER OF PROBLEM
C  N2=2*N = NO. OF COLUMNS IN W
C  X = INITIAL GUESS FOR EIGENVECTOR - SHOULD NOT BE NULL
C  Y = WORKING VECTOR
C  X & Y OF LENGTH N AT LEAST
C  SHIFT = SHIFT TO TRANSFORM PROBLEM TO ONE WITH EV CLOSEST TO SHIFT
C    WITH EV CLOSEST TO SHIFT
C  EPS=MACHINE PRECISION--SMALLEST NUMBER S.T. 1.0+EPS.GT.1.0
C  LIMIT=UPPER BOUND TO NUMBER OF ITERATIONS
C       = ON OUTPUT THE NUMBER OF ITERATIONS USED
C  EV=EIGENVALUE CALCULATED
C  IPR=PRINT CHANNEL. IPR=0 SUPPRESSES PRINTING.
C  STEP 0
    INTEGER N,N2,NW,LIMIT,COUNT,I,J,JN,K,N1,I1
    REAL W(NW,N2),X(N),Y(N),EPS,SHIFT,EV,S,T,P
C  SAFETY CHECK
    IF(N2.NE.2*N)STOP
C  STEP 1
    T=0.0
    DO 10 I=1,N
      Y(I)=0.0
      S=0.0
      DO 5 J=1,N
        JN=J+N
        W(I,J)=W(I,J)-SHIFT*W(I,JN)
        S=S+ABS(W(I,J))
  5   CONTINUE
      IF(T.LT.S)T=S
 10 CONTINUE
    T=T*EPS
C  STEP 2
    N1=N-1
    DO 100 I=1,N1
C  STEP 3
      S=ABS(W(I,I))
      K=I
```

```
        I1=I+1
        DO 20 J=I1,N
          IF(ABS(W(J,I)).LE.S)GOTO 20
          S=ABS(W(J,I))
          K=J
 20     CONTINUE
      IF(S.GT.0.0)GOTO 30
C  STEP 4
        W(I,I)=T
        GOTO 100
C  STEP 5
 30     IF(K.EQ.I)GOTO 50
C  STEP 6
        DO 40 J=I,N2
          S=W(I,J)
          W(I,J)=W(K,J)
          W(K,J)=S
 40     CONTINUE
C  STEP 7
 50     DO 80 J=I1,N
          S=W(J,I)/W(I,I)
          DO 70 K=I,N2
            W(J,K)=W(J,K)-S*W(I,K)
 70       CONTINUE
 80     CONTINUE
C  STEP 8
100   CONTINUE
C  STEP 9
      IF(ABS(W(N,N)).EQ.0.0)W(N,N)=T
C  STEP 10
      COUNT=0
C  STEP 11
110   COUNT=COUNT+1
      M=N
      S=X(N)
      X(N)=Y(N)
      Y(N)=S/W(N,N)
      P=ABS(Y(N))
C  STEP 12
      DO 130 JN=1,N1
        I=N-JN
        S=X(I)
        X(I)=Y(I)
        I1=I+1
        DO 120 J=I1,N
          S=S-W(I,J)*Y(J)
120     CONTINUE
        Y(I)=S/W(I,I)
        IF(ABS(Y(I)).LE.P)GOTO 130
        P=ABS(Y(I))
        M=I
130   CONTINUE
C  STEP 13
```

```
      EV=SHIFT+X(M)/Y(M)
C  STEP 14
      P=Y(M)
      M=0
      DO 140 I=1,N
        Y(I)=Y(I)/P
        IF(FLOAT(N)+Y(I).EQ.FLOAT(N)+X(I))M=M+1
 140  CONTINUE
      IF(IPR.GT.0)WRITE(IPR,960)COUNT,EV,M
 960  FORMAT(14H ITERATION NO.,I4,14H   APPROX. EV=,1PE16.8,5X,I4,
     *27H VECTOR ELEMENTS TEST EQUAL)
C  STEP 15
      IF(M.EQ.N)GOTO 200
      IF(COUNT.GT.LIMIT)GOTO 200
C  STEP 16
      DO 160 I=1,N
        S=0.0
        DO 150 J=1,N
          JN=J+N
          S=S+W(I,JN)*Y(J)
 150    CONTINUE
        X(I)=S
 160  CONTINUE
C  STEP 17
      GOTO 110
 200  LIMIT=COUNT
      RETURN
      END
```

Example output

We illustrate by finding a single eigensolution of the Hilbert segments of order 5 and 10. ?? Do we want to swap in the Frank matrix (the computations are generally easier)?

```
## #!/bin/bash
gfortran ../fortran/a10.f
mv ./a.out ../fortran/a10.run
../fortran/a10.run
```

```
##  ORDER=   5
##  USING SHIFT OF     0.000000
##  CONVERGED TO EV=  3.29019417E-06  IN    5  ITERATIONS
##  X(  1)= -0.80475118E-02
##  X(  2)=  0.15210588E+00
##  X(  3)= -0.65976608E+00
##  X(  4)=  0.10000000E+01
##  X(  5)= -0.49041715E+00
##  RESIDUAL(  1)= -0.74505806E-08
##  RESIDUAL(  2)=  0.00000000E+00
##  RESIDUAL(  3)= -0.74505806E-08
##  RESIDUAL(  4)= -0.37252903E-08
##  RESIDUAL(  5)= -0.37252903E-08
##  ORDER=  10
##  USING SHIFT OF     0.000000
##  CONVERGED TO EV=  1.26338462E-09  IN  101  ITERATIONS
```

```
##  X(  1)=  0.50510102E-05
##  X(  2)= -0.61139709E-03
##  X(  3)=  0.65672603E-02
##  X(  4)= -0.65278080E-02
##  X(  5)= -0.94817474E-01
##  X(  6)=  0.25418818E+00
##  X(  7)=  0.62985711E-01
##  X(  8)= -0.86295480E+00
##  X(  9)=  0.10000000E+01
##  X( 10)= -0.35897413E+00
##  RESIDUAL(  1)=  0.00000000E+00
##  RESIDUAL(  2)=  0.00000000E+00
##  RESIDUAL(  3)= -0.18626451E-08
##  RESIDUAL(  4)= -0.11175871E-07
##  RESIDUAL(  5)= -0.37252903E-08
##  RESIDUAL(  6)=  0.18626451E-08
##  RESIDUAL(  7)= -0.18626451E-08
##  RESIDUAL(  8)= -0.18626451E-08
##  RESIDUAL(  9)= -0.37252903E-08
##  RESIDUAL( 10)=  0.37252903E-08
```

BASIC

Listing

```
5 DIM A(10, 20),X(10),Y(10)
10 PRINT "GII JULY 25 77 ALG 10"
20 PRINT "GAUSS ELIMINATION FOR INVERSE ITERATION"
30 PRINT "ORDER=",
40 READ N
50 PRINT N
55 IF N <= 0 THEN QUIT : REM BWBASIC VARIANT
60 GOSUB 1500: REM BUILD OR INPUT MATRIX
70 GOSUB 2000: REM PUT METRIC IN RIGHT HALF OF A
75 GOSUB 1000: REM INITIAL GUESS TO VECTOR
80 LET K9=0 : REM SHIFT OF 0 FOR THIS EXAMPLE
90 PRINT "SHIFT=",K9
95 LET E9=K9
100 REM PRINT
105 LET T2=N: REM FACTOR FOR CONVERGENCE TEST
110 LET T1=0: REM STEP 1
120 FOR I=1 TO N
130 LET Q=0
140 FOR J=1 TO N
150 LET A(I,J)=A(I,J)-K9*A(I,J+N)
160 LET S=S+ABS(A(I,J))
170 NEXT J
180 IF T1>=S THEN GOTO 200
190 LET T1=S
200 NEXT I
205 LET T1=T1*1.0E-7: REM NS 8 DIGIT BASIC
210 FOR I=1 TO N-1: REM STEP 2
218 LET S=ABS(A(I,I)): REM STEP 3
226 LET K=I
```

```
234 FOR J=I+1 TO N
242 IF ABS(A(J,I))<=S THEN GOTO 266
250 LET S=ABS(A(J,I))
258 LET K=J
266 NEXT J
274 IF S>0 THEN GOTO 298: REM STEP 4
282 LET A(I,I)=T1
290 GOTO 394
298 IF K=I THEN GOTO 346: REM STEP 5
306 FOR J=I TO 2*N: REM STEP 6
314 LET S=A(I,J)
322 LET A(I,J)=A(K,J)
330 LET A(K,J)=S
338 NEXT J
346 FOR J=I+1 TO N: REM STEP 7
354 LET S=A(J,I)/A(I,I)
362 FOR K=I TO 2*N
370 LET A(J,K)=A(J,K)-S*A(I,K)
378 NEXT K
386 NEXT J
394 NEXT I: REM STEP 8
402 IF ABS(A(N,N))>0 THEN GOTO 420: REM STEP 9
410 LET A(N,N)=T1
420 LET I9=0: REM STEP 10
430 LET I9=I9+1: REM STEP 11
440 LET M=N
445 LET S=X(N)
450 LET X(N)=Y(N)
455 LET Y(N)=S/A(N,N)
460 LET P=ABS(Y(N))
470 FOR I=(N-1) TO 1 STEP -1: REM STEP 12
480 LET S=X(I)
485 LET X(I)=Y(I)
490 FOR J=I+1 TO N
500 LET S=S-A(I,J)*Y(J)
510 NEXT J
520 LET Y(I)=S/A(I,I)
530 IF ABS(Y(I))<=P THEN GOTO 560
540 LET M=I
550 LET P=ABS(Y(I))
560 NEXT I
570 LET E8=K9+X(M)/Y(M): REM STEP 13
580 REM PRINT "APPROX EV=",E8
600 LET P=Y(M): REM STEP 14
610 LET M=0
620 FOR I=1 TO N
630 LET Y(I)=Y(I)/P
635 IF T2+Y(I)<>T2+X(I) THEN GOTO 640
636 LET M=M+1
640 NEXT I
644 IF M=N THEN GOTO 730: REM STEP 15 -- CONVERGENCE TEST
645 IF I9>100 THEN GOTO 730: REM LIMIT SET AT 100
650 FOR I=1 TO N: REM STEP 16
```

```
660 LET S=0
670 FOR J=1 TO N
680 LET S=S+A(I,J+N)*Y(J)
690 NEXT J
700 LET X(I)=S
710 NEXT I
720 GOTO 430: REM STEP 17
725 REM STEP 18 -- END AND RESIDUALS
730 PRINT "CONVERGED TO EV=",E8," IN ",I9," ITNS"
735 PRINT M," EQUAL CPNTS IN VECTOR BETWEEN ITERATIONS"
740 GOSUB 1500: REM GET MATRIX AGAIN
750 GOSUB 2000: REM GET METRIC AGAIN
755 LET S=0: REM COMPUTE VECTOR INNER PRODUCT
760 FOR I=1 TO N
770 FOR J=1 TO N
780 LET S=S+Y(I)*A(I,J+N)*Y(J)
790 NEXT J
800 NEXT I
810 LET S=1/SQR(S)
815 PRINT "VECTOR"
820 FOR I=1 TO N
830 LET Y(I)=Y(I)*S: REM VECTOR NORMALIZATION
840 PRINT Y(I);
845 IF I=5*INT(I/5) THEN PRINT
850 NEXT I
855 PRINT
860 PRINT "RESIDUALS"
870 FOR I=1 TO N
880 LET S=0
890 FOR J=1 TO N: REM MATRIX * VECTOR - VALUE * METRIC * VECTOR
900 LET S=S+(A(I,J)-E8*A(I,J+N))*Y(J)
910 NEXT J
920 PRINT S;
925 IF 5*INT(I/5)=I THEN PRINT
930 NEXT I
940 PRINT
950 GOTO 40 : REM NEXT TRY
960 DATA 5, 10, -1
1000 REM          INITIAL X
1010 FOR I=1 TO N
1020 LET X(I)=1: REM MAY BE A POOR CHOICE
1030 NEXT I
1040 RETURN
1500 REM A IN FOR FRANK MATRIX
1505 PRINT "FRANK MATRIX"
1510 FOR I=1 TO N
1520 FOR J=1 TO I
1530 A(I,J)=J
1540 A(J,I)=J
1550 NEXT J
1560 NEXT I
1570 RETURN
2000 REM UNIT B IN RIGHT HALF OF MATRIX
```

```
2010 FOR I=1 TO N
2020 FOR J=1 TO N
2030 A(I,J+N)=0
2040 NEXT J
2050 A(I,I+N)=1
2060 NEXT I
2070 RETURN
```

Example output

In this case we use the Frank matrix for our test.

```
bwbasic ../BASIC/a10.bas >../BASIC/a10.out
# echo "done"
```

```
Bywater BASIC Interpreter/Shell, version 2.20 patch level 2

Copyright (c) 1993, Ted A. Campbell

Copyright (c) 1995-1997, Jon B. Volkoff


GII JULY 25 77 ALG 10
GAUSS ELIMINATION FOR INVERSE ITERATION
ORDER=
 5
FRANK MATRIX
SHIFT=        0
CONVERGED TO EV=            0.2715541     IN            101            ITNS
 1            EQUAL CPNTS IN VECTOR BETWEEN ITERATIONS
FRANK MATRIX
VECTOR
 0.3260187 -0.5485287 0.5968848 -0.4557341 0.1698911

RESIDUALS
 -0 0 -0 -0 0

 10
FRANK MATRIX
SHIFT=        0
CONVERGED TO EV=            0.2556738     IN            101            ITNS
 1            EQUAL CPNTS IN VECTOR BETWEEN ITERATIONS
FRANK MATRIX
VECTOR
 0.1281224 -0.2449948 0.3403202 -0.405639 0.4350771
 -0.4258922 0.3787616 -0.2977698 0.1900823 -0.0653188

RESIDUALS
 -0.0000087 0.0000141 -0.0000143 0.000009 -0
 -0.0000097 0.0000166 -0.0000183 0.0000141 -0.0000053

 -1
```

Pascal

Listing

```
program dr10(input,output);
{dr10.pas == driver to use Gauss elimination for inverse iteration
calculation of matrix eigensolutions

          Copyright 1988 J.C.Nash
}
{constype.def ==
  This file contains various definitions and type statements which are
  used throughout the collection of "Compact Numerical Methods".  In many
  cases not all definitions are needed, and users with very tight memory
  constraints may wish to remove some of the lines of this file when
  compiling certain programs.

  Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}

const
  big = 1.0E+35;    {a very large number}
  Maxconst = 25;    {Maximum number of constants in data record}
  Maxobs = 100;     {Maximum number of observations in data record}
  Maxparm  = 25;    {Maximum number of parameters to adjust}
  Maxvars = 10;     {Maximum number of variables in data record}
  acctol = 0.0001;  {acceptable point tolerance for minimisation codes}
  maxm = 20;        {Maximum number or rows in a matrix}
  maxn = 20;        {Maximum number of columns in a matrix}
  maxmn = 40;       {maxn+maxm, the number of rows in a working array}
  maxsym = 210;     {maximum number of elements of a symmetric matrix
              which need to be stored = maxm * (maxm + 1)/2 }
  reltest = 10.0;   {a relative size used to check equality of numbers.
              Numbers x and y are considered equal if the
              floating-point representation of reltest+x equals
              that of reltest+y.}
  stepredn = 0.2;   {factor to reduce stepsize in line search}
  yearwrit = 1990;  {year in which file was written}

type
  str2 = string[2];
  rmatrix = array[1..maxm, 1..maxn] of real; {a real matrix}
  wmatrix = array[1..maxmn, 1..maxn] of real; {a working array, formed
                as one real matrix stacked on another}
  smatvec = array[1..maxsym] of real; {a vector to store a symmetric matrix
              as the row-wise expansion of its lower triangle}
  rvector = array[1..maxm] of real;  {a real vector. We will use vectors
              of m elements always. While this is NOT space efficient,
              it simplifies program codes.}
  cgmethodtype= (Fletcher_Reeves,Polak_Ribiere,Beale_Sorenson);
    {three possible forms of the conjugate gradients updating formulae}
  probdata = record
```

```
          m     : integer; {number of observations}
          nvar  : integer; {number of variables}
          nconst: integer; {number of constants}
          vconst: array[1..Maxconst] of real;
          Ydata : array[1..Maxobs, 1..Maxvars] of real;
          nlls  : boolean; {true if problem is nonlinear least squares}
        end;
{
  NOTE: Pascal does not let us define the work-space for the function
  within the user-defined code.  This is a weakness of Pascal for this
  type of work.

}
var {global definitions}
  banner     : string[80]; {program name and description}

function calceps:real;
{calceps.pas ==
  This function returns the machine EPSILON or floating point tolerance,
  the smallest positive real number such that 1.0 + EPSILON > 1.0.
  EPSILON is needed to set various tolerances for different algorithms.
  While it could be entered as a constant, I prefer to calculate it, since
  users tend to move software between machines without paying attention to
  the computing environment. Note that more complete routines exist.
}
var
  e,e0: real;
  i: integer;
begin {calculate machine epsilon}
  e0 := 1; i:=0;
  repeat
    e0 := e0/2; e := 1+e0;  i := i+1;
  until (e=1.0) or (i=50); {note safety check}
  e0 := e0*2;
{ Writeln('Machine EPSILON =',e0);}
  calceps:=e0;
end; {calceps}

function genevres(n: integer; {order of matrices}
              A, B: rmatrix;
              evalue : real; {eigenvalue}
              X : rvector; {presumed eigenvector}
              print: boolean) {true for printing}
              : real; {returns sum of squared residuals}

{genevres.pas

  -- to compute residuals of generalized (symmetric) matrix
  eigenvalue problem

        A x = evalue B x
}
```

```
var
  t, ss : real;
  i,j : integer;

begin
  if print then
  begin
    writeln('Generalized matrix eigensolution residuals');
  end;
  ss:=0.0; {to accumulate the sum of squared residuals}
  for i:=1 to n do
  begin
    t:=0.0;
    for j:=1 to n do
    t:=t+(A[i,j]-evalue*B[i,j])*X[j];
    if print then
    begin
      write(t:10,' ');
    if (7 * (i div 7) = i) and (i<n) then writeln;
    end; {if print}
    ss:=ss+t*t;
  end;
  if print then
  begin
    writeln;
    writeln('Sum of squared residuals =',ss);
  end; {if print}
  genevres:=ss; {return sum of squared residuals}
end; {genevres.pas == residuals for generalized eigenproblem}

function rayquo(n :integer;
            A,B : rmatrix;
            Y : rvector): real;
{rayquo.pas
  == compute Rayleigh quotient. If the denominator of
      the quotient is zero, set the quotient to a large
      negative number (-big)
}
var
  s,t : real;
  i,j : integer;

begin
  s:=0.0; t:=0.0;
  for i:=1 to n do
  begin
    for j:=1 to n do
    begin
      s:=s+Y[i]*A[i,j]*Y[j];
      t:=t+Y[i]*B[i,j]*Y[j];
    end; {loop on j}
  end; {loop on i}
  if t>0.0 then rayquo:=s/t else rayquo:=-big;
  {note the safety value for the result}
```

```
end; {rayquo.pas == compute Rayleigh quotient}

Procedure Frank2(var m, n: integer; var A: rmatrix);
var
  i,j: integer;
begin
    for i:=1 to m do
    begin
        for j:=1 to n do
        begin
          if (i <= j) then
             A[i,j]:=i
          else
             A[i,j]:=j;
        end;
    end;
end;

Procedure gelim( n : integer;
                 p : integer;
                var A : rmatrix;
                 tol : real);


var
  det, s : real;
  h,i,j,k: integer;

begin
  det := 1.0;
  writeln('alg05.pas -- Gauss elimination with partial pivoting');
  for j := 1 to (n-1) do
  begin
    s := abs(A[j,j]); k := j;
    for h := (j+1) to n do
    begin
      if abs(A[h,j])>s then
      begin
        s := abs(A[h,j]); k := h;
      end;
    end;
    if k<>j then
    begin
      writeln('Interchanging rows ',k,' and ',j);
      for i := j to (n+p) do
      begin
        s := A[k,i]; A[k,i] := A[j,i]; A[j,i] := s;
      end;
      det := -det;
    end;
    det := det*A[j,j];
    if abs(A[j,j])<tol then
    begin
```

```
      writeln('Matrix computationally singular -- pivot < ',tol);
      halt;
    end;
    for k := (j+1) to n do
    begin
      A[k,j] := A[k,j]/A[j,j];
      for i := (j+1) to (n+p) do
          A[k,i] := A[k,i]-A[k,j]*A[j,i];
    end;
    det := det*A[n,n];
    if abs(A[n,n])<tol then
    begin
      writeln('Matrix computationally singular -- pivot < ',tol);
      halt;
    end;
  end;
  writeln('Gauss elimination complete -- determinant = ',det);
end;
procedure gii(nRow : integer;
             var A : rmatrix;
             var Y : rvector;
             var shift : real;
             var itcount: integer);

var
  i, itlimit, j, m, msame, nRHS :integer;
  ev, s, t, tol : real;
  X : rvector;

begin
  itlimit:=itcount;
  nRHS:=nRow;
  tol:=Calceps;
  s:=0.0;
  for i:=1 to nRow do
  begin
    X[i]:=Y[i];
    Y[i]:=0.0;
    for j:=1 to nRow do
    begin
      A[i,j]:=A[i,j]-shift*A[i,j+nRow];
      s:=s+abs(A[i,j]);
    end;
  end;
  tol:=tol*s;
  gelim(nRow, nRHS, A, tol);
  itcount:=0;
  msame :=0;
  while (msame<nRow) and (itcount<itlimit) do
  begin
    itcount:=itcount+1;
    m:=nRow; s:=X[nRow];
    X[nRow]:=Y[nRow];
```

```
    if abs(A[nRow,nRow])<tol then Y[nRow]:=s/tol
                             else Y[nRow]:=s/A[nRow,nRow];
    t:=abs(Y[nRow]);
    for i:=(nRow-1) downto 1 do
    begin
      s:=X[i]; X[i]:=Y[i];
      for j:=(i+1) to nRow do
      begin
        s:=s-A[i,j]*Y[j];
      end;
      if abs(A[i,i])<tol then Y[i]:=s/tol else  Y[i]:=s/A[i,i];
      if abs(Y[i])>t then
      begin
        m:=i; t:=abs(Y[i]);
      end;
    end;
    ev:=shift+X[m]/Y[m];
(*    writeln('Iteration ',itcount,'  approx. ev=',ev);*)

    t:=Y[m]; msame:=0;
    for i:=1 to nRow do
    begin
      Y[i]:=Y[i]/t;
      if reltest+Y[i] = reltest+X[i] then msame:=msame+1;

    end;

    if msame<nRow then
    begin
      for i:=1 to nRow do
      begin
        s:=0.0;
        for j:=1 to nRow do s:=s+A[i,j+nRow]*Y[j];
        X[i]:=s;
      end;
    end;
  end;
  if itcount>=itlimit then itcount:=-itcount;
  shift:=ev;
end;

var
  A, Acopy, B, Bcopy : rmatrix;
  Y : rvector;
  i, itcount, j, n, nRow, nCol : integer;
  rq, ss, Shift : real;
  vectorOK : boolean;

begin
  banner:='dr10.pas -- inverse iteration via Gauss elimination';
  repeat
    write('order of problem (n) = '); readln(n);
    writeln(n);
```

```
    if (n > 0) then
    begin {main loop over examples}
      nRow:=n; nCol:=2*n; {store matrices in an array n by 2n}
      writeln('Provide the A matrix');
      Frank2(nRow, nCol, Acopy);
      writeln('A matrix');
      for i:=1 to n do
      begin
        for j:=1 to n do
          begin
          write(Acopy[i,j]:10:5,' ');
          if (7 * (j div 7) = j) and (j<n) then writeln;
        end;
        writeln;
      end;
      writeln('B matrix set to unit matrix');
      for i:=1 to n do
      begin
        for j:=1 to n do B[i,j]:=0.0;
        Bcopy[i,i]:=1.0;
      end;
      writeln('B matrix');
      for i:=1 to n do
      begin
        for j:=1 to n do
        begin
          write(Bcopy[i,j]:10:5,' ');
          if (7 * (j div 7) = j) and (j<n) then writeln;
        end;
        writeln;
      end;
      shift:=0.0; {rem initial and safety value for the eigenvalue shift}
      vectorOK:=true; {approximate eigenvector will be all 1s for example}
      for i:=1 to n do Y[i]:=1.0; {Set starting vector}
      repeat
      if (not vectorOK) then
      begin
        writeln('Need a starting vector for inverse iteration');
        halt;
      end;
      vectorOK:=true; {set flag to indicate a vector is now in place}
(*
    writeln('Enter a shift for eigenvalues ([cr] = ',shift,') ');
    write(' A value > 1E+30 halts execution. Entry = ');
*)
      write('shift=?');
      readln(shift);
      writeln(shift);
      if (not (shift > 1e30)) then
      begin
        for i:=1 to n do  {copy matrices into working matrices}
        begin
          for j:=1 to n do
```

```
          begin
            A[i,j]:=Acopy[i,j]; B[i,j]:=Bcopy[i,j];
            A[i,j+n]:=B[i,j]; {to provide work matrix for ALG10}
          end; {loop on j}
        end; {loop on i}
        itcount:=100; {rem fairly liberal bound}
        gii(n,  A , Y, shift, itcount);
        writeln;
        if itcount > 0 then
        begin
          writeln(
          'Converged to eigenvalue =',shift,'  in  ',itcount,' iterations');
        end
        else
        begin
          writeln('Not converged. Approximate eigenvalue=',shift,
                  ' after ',-itcount,' iterations');
        end; {else not converged}
        writeln('Eigenvector');
        for i:=1 to n do
        begin
          write(Y[i]:10:5,' ');
          if (7* (i div 7) = i) and (i<n) then  writeln;
        end; {loop on i}
        writeln;
        ss:=genevres(n, Acopy, Bcopy, shift, Y, false);
        rq:=rayquo(n, Acopy, Bcopy, Y);
        writeln('Rayleigh quotient = ',rq,'   sumsquared err=',ss:12:9);
      end; {if shift <=1e30}
    until (shift>1e30); {end of loop over possible shifts}
    end; {main loop over n block}
  until (n <= 0);
end. {dr10.pas}
```

Example output

```
fpc ../Pascal2021/dr10.pas
# copy to run file
mv ../Pascal2021/dr10 ../Pascal2021/dr10.run
../Pascal2021/dr10.run <../Pascal2021/dr10p.in >../Pascal2021/dr10p.out
```

```
## Free Pascal Compiler version 3.0.4+dfsg-23 [2019/11/25] for x86_64
## Copyright (c) 1993-2017 by Florian Klaempfl and others
## Target OS: Linux for x86-64
## Compiling ../Pascal2021/dr10.pas
## Linking ../Pascal2021/dr10
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 402 lines compiled, 0.1 sec
```

```
order of problem (n) = 5
Provide the A matrix
A matrix
   1.00000     1.00000     1.00000     1.00000     1.00000
   1.00000     2.00000     2.00000     2.00000     2.00000
```

```
   1.00000    2.00000    3.00000    3.00000    3.00000
   1.00000    2.00000    3.00000    4.00000    4.00000
   1.00000    2.00000    3.00000    4.00000    5.00000
B matrix set to unit matrix
B matrix
   1.00000    0.00000    0.00000    0.00000    0.00000
   0.00000    1.00000    0.00000    0.00000    0.00000
   0.00000    0.00000    1.00000    0.00000    0.00000
   0.00000    0.00000    0.00000    1.00000    0.00000
   0.00000    0.00000    0.00000    0.00000    1.00000
shift=? 0.0000000000000000E+000
alg05.pas -- Gauss elimination with partial pivoting
Gauss elimination complete -- determinant =  2.4000000000000000E+001

Not converged. Approximate eigenvalue= 2.7155412933904033E-001 after 100 iterations
Eigenvector
   0.54620   -0.91899    1.00000   -0.76352    0.28463
Rayleigh quotient =  2.7155412933882112E-001   sumsquared err= 0.000000000
shift=? 1.0000000000000001E+032
order of problem (n) = 10
Provide the A matrix
A matrix
   1.00000    1.00000    1.00000    1.00000    1.00000    1.00000    1.00000
   1.00000    1.00000    1.00000
   1.00000    2.00000    2.00000    2.00000    2.00000    2.00000    2.00000
   2.00000    2.00000    2.00000
   1.00000    2.00000    3.00000    3.00000    3.00000    3.00000    3.00000
   3.00000    3.00000    3.00000
   1.00000    2.00000    3.00000    4.00000    4.00000    4.00000    4.00000
   4.00000    4.00000    4.00000
   1.00000    2.00000    3.00000    4.00000    5.00000    5.00000    5.00000
   5.00000    5.00000    5.00000
   1.00000    2.00000    3.00000    4.00000    5.00000    6.00000    6.00000
   6.00000    6.00000    6.00000
   1.00000    2.00000    3.00000    4.00000    5.00000    6.00000    7.00000
   7.00000    7.00000    7.00000
   1.00000    2.00000    3.00000    4.00000    5.00000    6.00000    7.00000
   8.00000    8.00000    8.00000
   1.00000    2.00000    3.00000    4.00000    5.00000    6.00000    7.00000
   8.00000    9.00000    9.00000
   1.00000    2.00000    3.00000    4.00000    5.00000    6.00000    7.00000
   8.00000    9.00000   10.00000
B matrix set to unit matrix
B matrix
   1.00000    0.00000    0.00000    0.00000    0.00000    0.00000    0.00000
   0.00000    0.00000    0.00000
   0.00000    1.00000    0.00000    0.00000    0.00000    0.00000    0.00000
   0.00000    0.00000    0.00000
   0.00000    0.00000    1.00000    0.00000    0.00000    0.00000    0.00000
   0.00000    0.00000    0.00000
   0.00000    0.00000    0.00000    1.00000    0.00000    0.00000    0.00000
   0.00000    0.00000    0.00000
   0.00000    0.00000    0.00000    0.00000    1.00000    0.00000    0.00000
```

```
   0.00000    0.00000    0.00000
   0.00000    0.00000    0.00000    0.00000    0.00000    1.00000    0.00000
   0.00000    0.00000    0.00000
   0.00000    0.00000    0.00000    0.00000    0.00000    0.00000    1.00000
   0.00000    0.00000    0.00000
   0.00000    0.00000    0.00000    0.00000    0.00000    0.00000    0.00000
   1.00000    0.00000    0.00000
   0.00000    0.00000    0.00000    0.00000    0.00000    0.00000    0.00000
   0.00000    1.00000    0.00000
   0.00000    0.00000    0.00000    0.00000    0.00000    0.00000    0.00000
   0.00000    0.00000    1.00000
shift=? 0.0000000000000000E+000
alg05.pas -- Gauss elimination with partial pivoting
Gauss elimination complete -- determinant =  3.6288000000000000E+005

Not converged. Approximate eigenvalue= 2.5567344134720177E-001 after 100 iterations
Eigenvector
   0.29440   -0.56298    0.78208   -0.93226    1.00000   -0.97898    0.87071
  -0.68457    0.43702   -0.15018
Rayleigh quotient =  2.5567965533013154E-001   sumsquared err= 0.000000009
shift=? 1.0000000000000001E+032
order of problem (n) = 0
```

Algorithms 11 and 12 – standardization and residuals for a complex eigensolution

These algorithms are probably among the least used of those included in Nashlib. Their intent was to allow proposed eigensolutions of complex matrices to be standardized and tested. This seemed a potentially important task in the 1970s. ?? include COMEIG (ref to Eberlein's work, others??)

The purpose of standardization is to facilitate comparisons between eigenvectors that are supposedly equivalent. Any (preferably) unit-length multiple of an eigenvector is also an eigenvector, so it is difficult to compare two proposed solutions for the same eigenvalue. Therefore we choose a multiplier so that the largest magnitude component of the eigenvector is set to $1 + 0i$ where

$$i = \sqrt{-1}$$

Fortran

The Algorithm 11 and 12 code:

```
C&&& A1112
C  TEST ALGORITHMS 11 & 12
C  J.C. NASH   JULY 1978, APRIL 1989
      INTEGER N,NS,NIN,NOUT
      REAL A(10,10),Z(10,10),X(10),Y(10),U(10),V(10),E,F,VNORM
      NA=10
      NIN=5
      NOUT=6
 10   READ(NIN,905)N
 905  FORMAT(I5)
      WRITE(NOUT,965)N
 965  FORMAT(' ORDER=',I4)
      IF(N.LE.0)STOP
C  READ MATRICES
      DO 20 I=1,N
        READ(NIN,906)(A(I,J),J=1,N)
 906    FORMAT(6F10.5)
 20   CONTINUE
      WRITE(NOUT,966)
 966  FORMAT(' MATRIX A')
      CALL OUT(A,NA,N,N,NOUT)
      DO 30 I=1,N
        READ(NIN,906)(Z(I,J),J=1,N)
 30   CONTINUE
      WRITE(NOUT,967)
 967  FORMAT(' MATRIX Z')
      CALL OUT(Z,NA,N,N,NOUT)
 40   READ(NIN,907)NS,E,F
 907  FORMAT(I5,5X,2F10.5)
      WRITE(NOUT,968)NS,E,F
 968  FORMAT(' SOLUTION',I4,'  EV=',1PE16.8,' + SQRT(-1)*',E16.8)
      IF(NS.LE.0)GOTO 10
      READ(NIN,906)(X(J),J=1,N)
      WRITE(NOUT,969)
 969  FORMAT(' REAL PART')
      WRITE(NOUT,970)(X(J),J=1,N)
 970  FORMAT(1H ,6E16.8)
```

```
      READ(NIN,906)(Y(J),J=1,N)
      WRITE(NOUT,971)
 971  FORMAT(' IMAGINARY PART')
      WRITE(NOUT,970)(Y(J),J=1,N)
      CALL A11VS(N,X,Y,VNORM)
      WRITE(NOUT,972)VNORM
 972  FORMAT(' STANDARDIZED VECTOR - NORM=',1PE16.8)
      WRITE(NOUT,969)
      WRITE(NOUT,970)(X(J),J=1,N)
      WRITE(NOUT,971)
      WRITE(NOUT,970)(Y(J),J=1,N)
      WRITE(NOUT,973)
 973  FORMAT(' RESIDUALS')
      CALL A12CVR(N,X,Y,A,10,Z,10,E,F,U,V)
      WRITE(NOUT,969)
      WRITE(NOUT,970)(U(J),J=1,N)
      WRITE(NOUT,971)
      WRITE(NOUT,970)(V(J),J=1,N)
      GOTO 40
      END
      SUBROUTINE A11VS(N,X,Y,VNORM)
C  ALGORITHM 11  -  COMPLEX VECTOR STANDARDISATION
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  STANDARDISES COMPEX VECTOR (N ELEMENTS)  X+SQRT(-1)*Y
C      TO  1.0+SQRT(-1)*0.0
C  VNORM = NORM OF VECTOR (LARGEST ELEMENT)
C  STEP 0
      INTEGER N,I,K
      REAL X(N),Y(N),VNORM,G,B,E,S
C  STEP 1
      G=0.0
C  STEP 2
      DO 60 I=1,N
C  STEP 3
        B=X(I)**2+Y(I)**2
C  STEP 4
        IF(B.LE.G)GOTO 60
C  STEP 5
        K=I
        G=B
C  STEP 6
  60  CONTINUE
C  SAVE NORM
      VNORM=G
C  SAFETY CHECK
      IF(G.EQ.0.0)RETURN
C  STEP 7
      E=X(K)/G
      S=-Y(K)/G
C  STEP 8
      DO 85 I=1,N
        G=X(I)*E-Y(I)*S
        Y(I)=Y(I)*E+X(I)*S
```

```
        X(I)=G
  85  CONTINUE
C  END
      RETURN
      END
      SUBROUTINE A12CVR(N,X,Y,A,NA,Z,NZ,E,F,U,V)
C  ALGORITHM 12 RESIDUALS OF A COMPLEX EIGENSOLUTION
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  N     =  ORDER OF PROBLEM
C  U + I*V =  ( A + I*Z - E - I*F)*(X + I*Y)  WHERE I=SQRT(-1)
C  NA,NZ = FIRST DIMENSIONS OF A & Z RESPECTIVELY
C  STEP 0
      INTEGER N,NA,NZ,J,K
      REAL A(NA,N),Z(NZ,N),X(N),Y(N),E,F,U(N),V(N),S,G
C  STEP 1
      DO 50 J=1,N
C  STEP 2
        S=-E*X(J)+F*Y(J)
        G=-F*X(J)-E*Y(J)
C  STEP 3
        DO 35 K=1,N
          S=S+A(J,K)*X(K)-Z(J,K)*Y(K)
          G=G+A(J,K)*Y(K)+Z(J,K)*X(K)
  35    CONTINUE
C  STEP 4  NOTE SAVE IN U & V
        U(J)=S
        V(J)=G
C  STEP 5
  50  CONTINUE
      RETURN
      END
      SUBROUTINE OUT(A,NA,N,NP,NOUT)
C  J.C. NASH   JULY 1978, APRIL 1989
      INTEGER NA,N,NOUT,I,J
      REAL A(NA,N)
      DO 20 I=1,N
        WRITE(NOUT,951)I
 951    FORMAT(' ROW',I3)
        WRITE(NOUT,952)(A(I,J),J=1,NP)
 952    FORMAT(1H ,1P5E16.8)
  20  CONTINUE
      RETURN
      END
```

Example output

We illustrate by finding a single eigensolution of the Hilbert segments of order 5 and 10. ?? Do we want to swap in the Frank matrix (the computations are generally easier)?

```
## #!/bin/bash
gfortran ../fortran/a1112.f
mv ./a.out ../fortran/a1112.run
../fortran/a1112.run <../fortran/a1112.in
```

```
##  ORDER=   2
```

```
## MATRIX A
## ROW  1
##   1.00000000E+00  0.00000000E+00
## ROW  2
##   0.00000000E+00  1.00000000E+00
## MATRIX Z
## ROW  1
##   0.00000000E+00 -1.00000000E+00
## ROW  2
##   1.00000000E+00  0.00000000E+00
## SOLUTION   1  EV=  2.00000000E+00 + SQRT(-1)*  0.00000000E+00
## REAL PART
##   0.50000000E+00  0.00000000E+00
## IMAGINARY PART
##   0.00000000E+00  0.50000000E+00
## STANDARDIZED VECTOR - NORM=  2.50000000E-01
## REAL PART
##   0.10000000E+01  0.00000000E+00
## IMAGINARY PART
##   0.00000000E+00  0.10000000E+01
## RESIDUALS
## REAL PART
##   0.00000000E+00  0.00000000E+00
## IMAGINARY PART
##   0.00000000E+00  0.00000000E+00
## SOLUTION   2  EV=  0.00000000E+00 + SQRT(-1)*  0.00000000E+00
## REAL PART
##  -0.10000000E+01  0.00000000E+00
## IMAGINARY PART
##   0.00000000E+00  0.10000000E+01
## STANDARDIZED VECTOR - NORM=  1.00000000E+00
## REAL PART
##   0.10000000E+01  0.00000000E+00
## IMAGINARY PART
##   0.00000000E+00 -0.10000000E+01
## RESIDUALS
## REAL PART
##   0.00000000E+00  0.00000000E+00
## IMAGINARY PART
##   0.00000000E+00  0.00000000E+00
## SOLUTION  -1  EV=  0.00000000E+00 + SQRT(-1)*  0.00000000E+00
## ORDER=   2
## MATRIX A
## ROW  1
##   1.00000000E+00  1.00000000E+00
## ROW  2
##   1.00000000E+00  1.00000000E+00
## MATRIX Z
## ROW  1
##   0.00000000E+00 -1.00000000E+00
## ROW  2
##   1.00000000E+00  0.00000000E+00
## SOLUTION   1  EV=  2.41420007E+00 + SQRT(-1)*  0.00000000E+00
## REAL PART
```

```
##    0.14141999E+01  0.10000000E+01
##  IMAGINARY PART
##    0.00000000E+00  0.10000000E+01
##  STANDARDIZED VECTOR - NORM=  2.00000000E+00
##  REAL PART
##    0.70709997E+00  0.10000000E+01
##  IMAGINARY PART
##   -0.70709997E+00  0.00000000E+00
##  RESIDUALS
##  REAL PART
##    0.19133091E-04 -0.23841858E-06
##  IMAGINARY PART
##   -0.19133091E-04  0.00000000E+00
##  SOLUTION   2  EV= -5.85799992E-01 + SQRT(-1)*  0.00000000E+00
##  REAL PART
##   -0.14141999E+01  0.10000000E+01
##  IMAGINARY PART
##    0.00000000E+00  0.10000000E+01
##  STANDARDIZED VECTOR - NORM=  2.00000000E+00
##  REAL PART
##   -0.70709997E+00  0.10000000E+01
##  IMAGINARY PART
##    0.70709997E+00  0.00000000E+00
##  RESIDUALS
##  REAL PART
##   -0.12131917E+00  0.17160004E+00
##  IMAGINARY PART
##    0.12131917E+00  0.00000000E+00
##  SOLUTION   0  EV=  0.00000000E+00 + SQRT(-1)*  0.00000000E+00
##  ORDER=   3
##  MATRIX A
##  ROW  1
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW  2
##    4.00000000E+00  5.00000000E+00  6.00000000E+00
##  ROW  3
##    7.00000000E+00  8.00000000E+00  9.00000000E+00
##  MATRIX Z
##  ROW  1
##    9.00000000E+00  8.00000000E+00  7.00000000E+00
##  ROW  2
##    6.00000000E+00  5.00000000E+00  4.00000000E+00
##  ROW  3
##    3.00000000E+00  2.00000000E+00  1.00000000E+00
##  SOLUTION   1  EV=  1.36844997E+01 + SQRT(-1)*  1.36844997E+01
##  REAL PART
##    0.49906299E+00  0.56529301E+00  0.63152498E+00
##  IMAGINARY PART
##    0.47247899E+00  0.10070200E+00 -0.27107400E+00
##  STANDARDIZED VECTOR - NORM=  4.72304910E-01
##  REAL PART
##    0.39612967E+00  0.69806379E+00  0.10000000E+01
##  IMAGINARY PART
##    0.91818923E+00  0.45909339E+00  0.00000000E+00
```

```
##  RESIDUALS
##  REAL PART
##   -0.69141388E-04  0.45776367E-04  0.16307831E-03
##  IMAGINARY PART
##    0.25606155E-03  0.20432472E-03  0.89049339E-04
##  SOLUTION   2  EV=  1.31529999E+00 + SQRT(-1)*  1.31531000E+00
##  REAL PART
##    0.74067497E+00 -0.24551900E-01 -0.78978002E+00
##  IMAGINARY PART
##   -0.27984101E+00 -0.11183600E+00  0.56165900E-01
##  STANDARDIZED VECTOR - NORM=  6.26910388E-01
##  REAL PART
##    0.10000000E+01  0.20914188E-01 -0.95817167E+00
##  IMAGINARY PART
##    0.00000000E+00 -0.14309023E+00 -0.28618470E+00
##  RESIDUALS
##  REAL PART
##    0.27894974E-04  0.14424324E-04 -0.43809414E-05
##  IMAGINARY PART
##    0.67234039E-04  0.22649765E-04 -0.16033649E-04
##  SOLUTION   3  EV=  0.00000000E+00 + SQRT(-1)*  0.00000000E+00
##  REAL PART
##   -0.40955499E+00  0.81911302E+00 -0.40955701E+00
##  IMAGINARY PART
##   -0.16320599E-01  0.32640599E-01 -0.16320400E-01
##  STANDARDIZED VECTOR - NORM=  6.72011554E-01
##  REAL PART
##   -0.49999815E+00  0.10000000E+01 -0.50000060E+00
##  IMAGINARY PART
##   -0.43958426E-06 -0.37252903E-08 -0.96857548E-07
##  RESIDUALS
##  REAL PART
##    0.47311187E-05  0.68247318E-05  0.91567636E-05
##  IMAGINARY PART
##    0.11682510E-04  0.64373016E-05  0.89406967E-06
##  SOLUTION  -1  EV=  0.00000000E+00 + SQRT(-1)*  0.00000000E+00
##  ORDER=   0
```

Pascal

?? Currently we do not seem to have an example driver for these two codes.

Listing – a11.pas

```
procedure stdceigv(n: integer;
                var T, U: rmatrix);

var
  i, k, m : integer;
  b, e, g, s : real;

begin
  writeln('alg11.pas -- standardized eigensolutions');
  writeln(confile,'alg11.pas -- standardized eigensolutions');
```

```
  for i := 1 to n do
  begin
    g := T[1,i]*T[1,i]+U[1,i]*U[1,i];

    k := 1;
    if n>1 then
    begin
      for m := 2 to n do
      begin
        b := T[m,i]*T[m,i]+U[m,i]*U[m,i];
        if b>g then
        begin
          k := m;
          g := b;
        end;
      end;
    end;
    e := T[k,i]/g;
    s := -U[k,i]/g;
    for k := 1 to n do
    begin
      g := T[k,i]*e-U[k,i]*s; U[k,i] := U[k,i]*e+T[k,i]*s; T[k,i] := g;
    end;
  end;
end;
```

Listing – a12.pas

```
procedure comres( i, n: integer;
                  A, Z, T, U, Acopy, Zcopy : rmatrix);

var
  j, k: integer;
  g, s, ss : real;

begin
  writeln('alg12.pas -- complex eigensolution residuals');
  writeln(confile,'alg12.pas -- complex eigensolution residuals');
  ss := 0.0;
  for j := 1 to n do
  begin
    s := -A[i,i]*T[j,i]+Z[i,i]*U[j,i]; g := -Z[i,i]*T[j,i]-A[i,i]*U[j,i];

    for k := 1 to n do
    begin
      s := s+Acopy[j,k]*T[k,i]-Zcopy[j,k]*U[k,i];
      g := g+Acopy[j,k]*U[k,i]+Zcopy[j,k]*T[k,i];
    end;
    writeln('(',s,',',g,')'); writeln(confile,'(',s,',',g,')');
    ss := ss+s*s+g*g;
  end;
  writeln('Sum of squares = ',ss); writeln(confile,'Sum of squares = ',ss);
  writeln; writeln(confile);
```

```
end;
```

Algorithm 13

Fortran

Listing

```
C&&& A13
C  TEST ALG. 13    JULY 1978
C  J.C. NASH   JULY 1978, APRIL 1989
      REAL H,EPS
      INTEGER N,ND,I,NOUT,NIN
      REAL A(10,10),B(10,10),AT(10,10),Z(10),V(10,10),RMAX,VMAX
      EXTERNAL FRANKM,UNITM
      ND=10
C  I/O CHANNELS
      NIN=5
      NOUT=6
   1  READ(NIN,900)N
 900  FORMAT(I4)
      WRITE(NOUT,901)N
 901  FORMAT(' ORDER N=',I4)
      IF(N.LE.0)STOP
      CALL FRANKM(N,N,V,ND)
      ISWP=30
C   IBM  SHORT PRECISION
      EPS=16.0**(-5)
C  IBM VALUE FOR BIG NO.
C&&&       H=R1MACH(2)
      H = 1.0E+35
      CALL A13ESV(N,V,ND,EPS,H,ISWP,NOUT,Z)
      WRITE(NOUT,903)ISWP
 903  FORMAT(' CONVERGED IN ',I4,' SWEEPS')
      CALL EVT(N,V,ND,Z,FRANKM,UNITM,AT,ND,B,ND,NOUT,RMAX,VMAX)
      GOTO 1
      END
      SUBROUTINE A13ESV(N,A,NA,EPS,H,ISWP,IPR,Z)
C  ALGORITHM 13 EIGENPROBLEM OF A REAL SYMMETRIC MATRIX VIA SVD
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  N     =  ORDER OF PROBLEM
C  A     =  ARRAY CONTAINING MATRIX FOR WHICH EIGENVALUES ARE TO BE
C           COMPUTED.  RETURNS EIGENVECTORS AS COLUMNS
C  NA    =  FIRST DIMENSION OF A
C  EPS   =  MACHINE PRECISION
C  H     =  A NUMBER LARGER THAN ANY POSSIBLE EIGENVALUE. CHANGED
C           DURING EXECUTION. DO NOT ENTER AS A CONSTANT
C  ISWP  =  LIMIT ON SWEEPS (INPUT). SWEEPS USED (OUTPUT).
C  IPR   =  PRINT CHANNEL. IPR.GT.0 FOR PRINTING.
C  Z     =  EIGENVALUES (OUTPUT)
C  STEP 0
      INTEGER N,NA,ISWP,IPR,LISWP,I,J,COUNT,N1,J1
      REAL A(NA,N),EPS,H,V,Z(N),P,Q,R,S,C
      LISWP=ISWP
      ISWP=0
      N1=N-1
```

```
C  STEP 1
      DO 5 I=1,N
        V=A(I,I)
        DO 3 J=1,N
          IF(J.EQ.I)GOTO 3
          V=V-ABS(A(I,J))
   3    CONTINUE
        IF(V.LT.H)H=V
   5  CONTINUE
      IF(H.LE.EPS)GOTO 6
      H=0.0
      GOTO 30
   6  H=H-SQRT(EPS)
C  STEP 2
      DO 15 I=1,N
        A(I,I)=A(I,I)-H
  15  CONTINUE
C  STEP 3
  30  COUNT=0
C  CHECK FOR ORDER 1 PROBLEMS AND SKIP WORK
      IF(N.EQ.1)GOTO 160
      ISWP=ISWP+1
      IF(ISWP.GT.LISWP)GOTO 160
C  STEP 4
      DO 140 J=1,N1
C  STEP 5
        J1=J+1
        DO 130 K=J1,N
C  STEP 6
          P=0.0
          Q=0.0
          R=0.0
          DO 65 I=1,N
            P=P+A(I,J)*A(I,K)
            Q=Q+A(I,J)**2
            R=R+A(I,K)**2
  65      CONTINUE
C  STEP 7
          IF(1.0.LT.1.0+ABS(P/SQRT(Q*R)))GOTO 80
          IF(Q.LT.R)GOTO 80
          COUNT=COUNT+1
          GOTO 130
  80      Q=Q-R
C  STEP 8
          V=SQRT(4.0*P*P+Q*Q)
          IF(V.EQ.0.0)GOTO 130
C  STEP 9
          IF(Q.LT.0.0)GOTO 110
C  STEP 10
          C=SQRT((V+Q)/(2.0*V))
          S=P/(V*C)
          GOTO 120
C  STEP 11
```

```
 110      S=SQRT((V-Q)/(2.0*V))
          IF(P.LT.0.0)S=-S
          C=P/(V*S)
C  STEP 12
 120      DO 125 I=1,N
            V=A(I,J)
            A(I,J)=V*C+A(I,K)*S
            A(I,K)=-V*S+A(I,K)*C
 125      CONTINUE
C  STEP 13
 130    CONTINUE
C  STEP 14
 140  CONTINUE
C  STEP 15
      IF(IPR.GT.0)WRITE(IPR,970)ISWP,COUNT
 970  FORMAT( 9H AT SWEEP,I4,2X,I4,18H ROTATIONS SKIPPED)
      IF(COUNT.LT.N*(N-1)/2)GOTO 30
C  STEP 16
 160  DO 168 J=1,N
        S=0.0
        DO 162 I=1,N
          S=S+A(I,J)**2
 162    CONTINUE
        S=SQRT(S)
        DO 164 I=1,N
          A(I,J)=A(I,J)/S
 164    CONTINUE
        R=S+H
        Z(J)=R
 168  CONTINUE
C  STEP 17
 170  RETURN
      END
      SUBROUTINE UNITM(M,N,A,NA)
C  PUTS UNIT MATRIX M BY N IN A
C  J.C. NASH   JULY 1978, APRIL 1989
      INTEGER M,N,NA,I,J
      REAL A(NA,N)
      DO 10 I=1,M
        DO 5 J=1,N
          A(I,J)=0.0
          IF(I.EQ.J)A(I,I)=1.0
   5    CONTINUE
  10  CONTINUE
      RETURN
      END
      SUBROUTINE EVT(N,V,NV,Z,AIN,BIN,A,NA,B,NB,NOUT,RMAX,VMAX)
C  J.C. NASH   JULY 1978, APRIL 1989
C  COMPUTES RESIDUALS AND INNER PRODUCTS
C    R  =  (A - Z(J)*B)*V(.,J)
C  AIN AND BIN ARE NAMES OF MATRIX CALCULATING ROUTINES FOR A AND B
C  WHOSE FIRST DIMENSIONS ARE NA AND NB RESP.
C  RMAX AND VMAX ARE MAX ABS RESIDUAL AND INNER PRODUCT RESP.
```

```
C
      INTEGER N,NV,NA,NB,NOUT,I,J,K,RPOSI,RPOSJ,VPOSI,VPOSJ,I1,N1
      REAL V(NV,N),A(NA,N),B(NB,N),Z(N),RMAX,VMAX
      DOUBLE PRECISION ACC,TACC,DABS,DBLE
      CALL AIN(N,N,A,NA)
      CALL BIN(N,N,B,NB)
      N1=N-1
      TACC=0.0
      RPOSI=1
      RPOSJ=1
      DO 20 I=1,N
        DO 15 J=1,N
          ACC=0.0
          DO 10 K=1,N
            ACC=ACC+DBLE(V(K,J))*(A(I,K)-Z(J)*B(I,K))
 10       CONTINUE
          IF(DABS(ACC).LE.TACC)GOTO 15
          TACC=DABS(ACC)
          RPOSI=I
          RPOSJ=J
 15     CONTINUE
 20   CONTINUE
      RMAX=TACC
      IF(NOUT.GT.0)WRITE(NOUT,951)RMAX,RPOSI,RPOSJ
951   FORMAT(' MAX. ABS. RESIDUAL=',1PE16.8,'  POSN',2I4)
      VPOSI=0
      VPOSJ=0
      TACC=0.0
      IF(N.EQ.1)GOTO 45
      DO 40 I=1,N1
        I1=I+1
        DO 35 J=I1,N
          ACC=0.0
          DO 30 K=1,N
            ACC=ACC+DBLE(V(K,I))*V(K,J)
 30       CONTINUE
          IF(DABS(ACC).LE.TACC)GOTO 35
          TACC=DABS(ACC)
          VPOSI=I
          VPOSJ=J
 35     CONTINUE
 40   CONTINUE
      VMAX=TACC
      IF(NOUT.GT.0)WRITE(NOUT,952)VMAX,VPOSI,VPOSJ
952   FORMAT(' MAX. ABS. INNER PRODUCT=',1PE16.8,'  POSN',2I4)
 45   IF(NOUT.LE.0)RETURN
      DO 50 J=1,N
        WRITE(NOUT,953)J,Z(J)
953   FORMAT(' EIGENVALUE',I3,'=',1PE16.8)
        WRITE(NOUT,954)(V(K,J),K=1,N)
954   FORMAT(1H ,5E16.8)
 50   CONTINUE
      RETURN
```

```
      END
      SUBROUTINE FRANKM(M,N,A,NA)
C  J.C. NASH   JULY 1978, APRIL 1989
      INTEGER M,N,NA,I,J
C  INPUTS FRANK MATRIX M BY N INTO A
      REAL A(NA,N)
      DO 20 I=1,M
        DO 10 J=1,N
          A(I,J)=AMIN0(I,J)
  10    CONTINUE
  20  CONTINUE
      RETURN
      END
```

Example output

```
## #!/bin/bash
gfortran ../fortran/dr13.f
mv ./a.out ../fortran/dr13.run
../fortran/dr13.run < ../fortran/dr13.in
```

```
##  ORDER N=   2
##  AT SWEEP   1     0 ROTATIONS SKIPPED
##  AT SWEEP   2     0 ROTATIONS SKIPPED
##  AT SWEEP   3     1 ROTATIONS SKIPPED
##  CONVERGED IN    3 SWEEPS
##  MAX. ABS. RESIDUAL=  1.47663954E-07  POSN   1   1
##  MAX. ABS. INNER PRODUCT=  0.00000000E+00  POSN   0   0
##  EIGENVALUE  1=  2.61803412E+00
##    0.52573115E+00  0.85065079E+00
##  EIGENVALUE  2=  3.81966025E-01
##   -0.85065079E+00  0.52573115E+00
##  ORDER N=   4
##  AT SWEEP   1     0 ROTATIONS SKIPPED
##  AT SWEEP   2     0 ROTATIONS SKIPPED
##  AT SWEEP   3     1 ROTATIONS SKIPPED
##  AT SWEEP   4     6 ROTATIONS SKIPPED
##  CONVERGED IN    4 SWEEPS
##  MAX. ABS. RESIDUAL=  6.81894505E-07  POSN   3   1
##  MAX. ABS. INNER PRODUCT=  4.60000820E-08  POSN   1   3
##  EIGENVALUE  1=  8.29086018E+00
##    0.22801343E+00  0.42852512E+00  0.57735032E+00  0.65653849E+00
##  EIGENVALUE  2=  1.00000048E+00
##   -0.57735056E+00 -0.57735002E+00  0.66493286E-07  0.57735020E+00
##  EIGENVALUE  3=  4.26022291E-01
##    0.65653813E+00 -0.22801332E+00 -0.57735056E+00  0.42852539E+00
##  EIGENVALUE  4=  2.83118486E-01
##   -0.42852521E+00  0.65653872E+00 -0.57735002E+00  0.22801307E+00
##  ORDER N=   4
##  AT SWEEP   1     0 ROTATIONS SKIPPED
##  AT SWEEP   2     0 ROTATIONS SKIPPED
##  AT SWEEP   3     1 ROTATIONS SKIPPED
##  AT SWEEP   4     6 ROTATIONS SKIPPED
##  CONVERGED IN    4 SWEEPS
```

```
## MAX. ABS. RESIDUAL= 6.81894505E-07 POSN 3 1
## MAX. ABS. INNER PRODUCT= 4.60000820E-08 POSN 1 3
## EIGENVALUE 1= 8.29086018E+00
##   0.22801343E+00 0.42852512E+00 0.57735032E+00 0.65653849E+00
## EIGENVALUE 2= 1.00000048E+00
##  -0.57735056E+00 -0.57735002E+00 0.66493286E-07 0.57735020E+00
## EIGENVALUE 3= 4.26022291E-01
##   0.65653813E+00 -0.22801332E+00 -0.57735056E+00 0.42852539E+00
## EIGENVALUE 4= 2.83118486E-01
##  -0.42852521E+00 0.65653872E+00 -0.57735002E+00 0.22801307E+00
## ORDER N= 10
## AT SWEEP 1    0 ROTATIONS SKIPPED
## AT SWEEP 2    0 ROTATIONS SKIPPED
## AT SWEEP 3    0 ROTATIONS SKIPPED
## AT SWEEP 4   32 ROTATIONS SKIPPED
## AT SWEEP 5   44 ROTATIONS SKIPPED
## AT SWEEP 6   42 ROTATIONS SKIPPED
## AT SWEEP 7   45 ROTATIONS SKIPPED
## CONVERGED IN   7 SWEEPS
## MAX. ABS. RESIDUAL= 2.39280362E-05 POSN 9 1
## MAX. ABS. INNER PRODUCT= 6.23372785E-08 POSN 6 10
## EIGENVALUE 1= 4.47660294E+01
##   0.65047376E-01 0.12864168E+00 0.18936241E+00 0.24585304E+00 0.29685175E+00
##   0.34121934E+00 0.37796459E+00 0.40626666E+00 0.42549327E+00 0.43521538E+00
## EIGENVALUE 2= 5.04890442E+00
##  -0.18936226E+00 -0.34121895E+00 -0.42549327E+00 -0.42549345E+00 -0.34121940E+00
##  -0.18936238E+00 -0.18430426E-06 0.18936227E+00 0.34121925E+00 0.42549381E+00
## EIGENVALUE 3= 1.87301636E+00
##   0.29685244E+00 0.43521363E+00 0.34121892E+00 0.65048993E-01 -0.24585134E+00
##  -0.42549297E+00 -0.37796640E+00 -0.12864257E+00 0.18936226E+00 0.40626761E+00
## EIGENVALUE 4= 9.99992371E-01
##  -0.37796077E+00 -0.37796682E+00 -0.31607331E-05 0.37796402E+00 0.37796602E+00
##   0.20208058E-05 -0.37796175E+00 -0.37796679E+00 -0.16900430E-05 0.37796503E+00
## EIGENVALUE 5= 6.43104553E-01
##   0.42549762E+00 0.18936004E+00 -0.34121791E+00 -0.34121671E+00 0.18935442E+00
##   0.42549407E+00 0.89485920E-05 -0.42549804E+00 -0.18936114E+00 0.34121749E+00
## EIGENVALUE 6= 4.65229034E-01
##  -0.43522137E+00 0.65052554E-01 0.42549083E+00 -0.12863764E+00 -0.40626609E+00
##   0.18935713E+00 0.37796399E+00 -0.24584912E+00 -0.34122151E+00 0.29685244E+00
## EIGENVALUE 7= 3.66199493E-01
##   0.40627682E+00 -0.29686981E+00 -0.18934317E+00 0.43520308E+00 -0.12863609E+00
##  -0.34122577E+00 0.37796903E+00 0.65047354E-01 -0.42548791E+00 0.24584727E+00
## EIGENVALUE 8= 3.07968140E-01
##  -0.34119982E+00 0.42547908E+00 -0.18935996E+00 -0.18936360E+00 0.42549238E+00
##  -0.34120587E+00 -0.21800142E-04 0.34124032E+00 -0.42551416E+00 0.18937302E+00
## EIGENVALUE 9= 2.73780823E-01
##   0.24583328E+00 -0.40622735E+00 0.42543337E+00 -0.29676920E+00 0.64941816E-01
##   0.18947297E+00 -0.37804705E+00 0.43525901E+00 -0.34123680E+00 0.12864587E+00
## EIGENVALUE 10= 2.55672455E-01
##  -0.12867406E+00 0.24592136E+00 -0.34131119E+00 0.40634301E+00 -0.43523723E+00
##   0.42545170E+00 -0.37787846E+00 0.29675686E+00 -0.18929419E+00 0.65023489E-01
## ORDER N=  1
## CONVERGED IN   0 SWEEPS
## MAX. ABS. RESIDUAL= 0.00000000E+00 POSN 1 1
```

```
## EIGENVALUE  1=  1.00000000E+00
##    0.10000000E+01
## ORDER N=   0
```

BASIC

Listing

```
10 PRINT "A13 EIGENSOLUTIONS OF A SYMMETRIC MATRIX VIA SVD"
20 LET N9=20
30 DIM A(N9,N9),V(N9,N9),B(N9,N9),D(N9),Z(N9)
40 PRINT "ORDER OF MATRIX: ";
50 READ N
55 PRINT N
60 IF (N <= 0) THEN QUIT
70 LET M=N
80 GOSUB 1500: REM BUILD FRANK MATRIX N BY N
100 GOSUB 3000
110 FOR I=1 TO N
120   PRINT "EV(";I;")=";D(I);"    RQ=";Z(I)
130   FOR J=1 TO N
140      PRINT V(J,I);" ";
150   NEXT J
155   PRINT
160 NEXT I
170 PRINT
200 GOTO 40
300 DATA  2, 4, 10, 1, 0
1500 REM PREPARE FRANK MATRIX IN A
1510 FOR I=1 TO M
1530 FOR J=1 TO N
1540 IF (I <= J) THEN LET A(I,J)=I ELSE LET A(I,J)=J
1545 LET B(I,J)=A(I,J)
1550 NEXT J
1560 NEXT I
1570 RETURN
3000 PRINT "SMEV.NJ ALG 13 DEC 7 78"
3002 LET E1=1E-7
3004 IF (N > 1) THEN GOTO 3020
3006 LET D(1)=A(1,1)
3008 LET V(1,1)=1
3010 LET Z(1)=D(1)
3012 RETURN
3014 REM END SPECIAL CASE N=1
3018 REM DIM A(N,N),V(N,N),B(N,N) (FOR RAYLEIGH QUOTIENT)
3020 LET H=1E+38: REM BIG NUMBER
3025 FOR I=1 TO N
3030   LET V1=A(I,I)
3035   FOR J=1 TO N
3040     LET B(I,J)=A(I,J)
3045     IF I=J THEN GOTO 3055
3050     LET V1=V1-ABS(A(I,J))
3055   NEXT J
3060   IF V1>=H THEN GOTO 3070
```

```
3065   LET H=V1
3070 NEXT I
3075 IF H<E1 THEN GOTO 3090
3080 LET H=0
3085 GOTO 3120
3090 LET H=H-SQR(E1)
3095 FOR I=1 TO N
3100   LET A(I,I)=A(I,I)-H
3105 NEXT I
3110 PRINT
3115 PRINT "SCALING BY SUBTRACTION OF",H
3120 LET I9=0
3125 LET N2=N*(N-1)/2
3130 LET N1=N-1
3135 LET N9=N2
3140 LET I9=I9+1
3145 LET N9=0
3150 IF I9<=30 THEN GOTO 3170
3155 PRINT
3160 PRINT "NON-",
3165 GOTO 3355
3170 FOR J=1 TO N1
3175   LET J1=J+1
3180   FOR K=J1 TO N
3185     LET P=0
3190     LET R=0
3195     LET Q=0
3200     FOR I=1 TO N
3205       LET P=P+A(I,J)*A(I,K)
3210       LET Q=Q+A(I,J)*A(I,J)
3215       LET R=R+A(I,K)*A(I,K)
3220     NEXT I
3225     IF 1+ABS(P/SQR(Q*R))>1 THEN GOTO 3235
3230     IF Q>=R THEN GOTO 3320
3235     LET Q=Q-R
3240     LET V1=SQR(4*P*P+Q*Q)
3245     IF V1=0 THEN GOTO 3320
3250     IF Q<0 THEN GOTO 3270
3255     LET C=SQR((V1+Q)/(2*V1))
3260     LET S=P/(V1*C)
3265     GOTO 3290
3270     LET S=SQR((V1-Q)/(2*V1))
3275     IF P>0 THEN GOTO 3285
3280     LET S=-S
3285     LET C=P/(V1*S)
3290     FOR I=1 TO N
3295       LET V1=A(I,J)
3300       LET A(I,J)=V1*C+A(I,K)*S
3305       LET A(I,K)=-V1*S+A(I,K)*C
3310     NEXT I
3315     GOTO 3325
3320     LET N9=N9+1
3325   NEXT K
```

```
3330 NEXT J
3335 IF N9=N2 THEN GOTO 3350
3340 PRINT "SWEEP",I9,"  ",N9,"SMALL P"
3345 GOTO 3140
3350 PRINT
3355 PRINT "CONVERGENCE AT SWEEP",I9
3360 LET V1=0
3365 LET C=0
3370 FOR J=1 TO N
3375   LET Q=0
3380   FOR I=1 TO N
3385     LET Q=Q+A(I,J)^2
3390   NEXT I
3395   LET Q=SQR(Q)
3400   FOR I=1 TO N
3405     LET V(I,J)=A(I,J)/Q
3410   NEXT I
3415   LET Q=Q+H
3420   GOSUB 3440
3425   LET D(J)=Q
3430 NEXT J
3435 RETURN
3440 LET Q=0
3445 FOR I=1 TO N
3450   FOR K=1 TO N
3455     LET Q=Q+V(I,J)*B(I,K)*V(K,J)
3460   NEXT K
3465 NEXT I
3470 LET Z(J)=Q
3475 RETURN
```

Example output

```
bwbasic ../BASIC/a13.bas  >../BASIC/a13.out
# echo "done"
```

```
Bywater BASIC Interpreter/Shell, version 2.20 patch level 2

Copyright (c) 1993, Ted A. Campbell

Copyright (c) 1995-1997, Jon B. Volkoff


A13 EIGENSOLUTIONS OF A SYMMETRIC MATRIX VIA SVD
ORDER OF MATRIX:  2
SMEV.NJ ALG 13 DEC 7 78

SCALING BY SUBTRACTION OF   -0.0003162
SWEEP         1                           0            SMALL P

CONVERGENCE AT SWEEP        2
EV( 1)= 2.6180340    RQ= 2.6180340
 0.5257311  0.8506508
EV( 2)= 0.381966    RQ= 0.381966
```

```
 -0.8506508  0.5257311

ORDER OF MATRIX:  4
SMEV.NJ ALG 13 DEC 7 78

SCALING BY SUBTRACTION OF   -3.0003162
SWEEP         1                          0            SMALL P
SWEEP         2                          0            SMALL P
SWEEP         3                          0            SMALL P
SWEEP         4                          1            SMALL P

CONVERGENCE AT SWEEP        5
EV( 1)= 8.2908594    RQ= 8.2908594
 0.2280134  0.428525  0.5773503  0.6565385
EV( 2)= 1    RQ= 1
 -0.5773503  -0.5773503  0  0.5773503
EV( 3)= 0.426022    RQ= 0.426022
 0.6565385  -0.2280134  -0.5773503  0.428525
EV( 4)= 0.2831186    RQ= 0.2831186
 -0.428525  0.6565385  -0.5773503  0.2280134

ORDER OF MATRIX:  10
SMEV.NJ ALG 13 DEC 7 78

SCALING BY SUBTRACTION OF   -36.0003162
SWEEP         1                          0            SMALL P
SWEEP         2                          0            SMALL P
SWEEP         3                          0            SMALL P
SWEEP         4                          0            SMALL P
SWEEP         5                          25           SMALL P

CONVERGENCE AT SWEEP        6
EV( 1)= 44.7660687    RQ= 44.7660687
 0.0650474  0.1286417  0.1893624  0.245853  0.2968517  0.3412192  0.3779645  0.
4062666  0.4254934  0.4352154
EV( 2)= 5.0489173    RQ= 5.0489173
 -0.1893624  -0.3412192  -0.4254934  -0.4254934  -0.3412192  -0.1893624  -0  0.
1893624  0.3412192  0.4254934
EV( 3)= 1.873023    RQ= 1.873023
 0.2968517  0.4352154  0.3412192  0.0650474  -0.245853  -0.4254934  -0.3779645
 -0.1286417  0.1893624  0.4062666
EV( 4)= 1    RQ= 1
 -0.3779645  -0.3779645  -0  0.3779645  0.3779645  0  -0.3779645  -0.3779645  0
  0.3779645
EV( 5)= 0.6431041    RQ= 0.6431041
 0.4254934  0.1893624  -0.3412192  -0.3412192  0.1893624  0.4254934  0  -0.4254
934  -0.1893624  0.3412192
EV( 6)= 0.465233    RQ= 0.465233
 -0.4352154  0.0650474  0.4254934  -0.1286417  -0.4062666  0.1893624  0.3779645
  -0.245853  -0.3412192  0.2968517
EV( 7)= 0.3662089    RQ= 0.3662089
 0.4062666  -0.2968517  -0.1893624  0.4352154  -0.1286417  -0.3412192  0.377964
5  0.0650474  -0.4254934  0.245853
```

```
EV( 8)= 0.3079785    RQ= 0.3079785
 -0.3412192  0.4254934  -0.1893624  -0.1893624  0.4254934  -0.3412192  -0  0.34
12192  -0.4254934  0.1893624
EV( 9)= 0.2737868    RQ= 0.2737868
 0.245853  -0.4062666  0.4254934  -0.2968517  0.0650474  0.1893624  -0.3779645
 0.4352154  -0.3412192  0.1286417
EV( 10)= 0.2556796    RQ= 0.2556796
 -0.1286417  0.245853  -0.3412192  0.4062666  -0.4352154  0.4254934  -0.3779645
  0.2968517  -0.1893624  0.0650474

ORDER OF MATRIX:  1
SMEV.NJ ALG 13 DEC 7 78
EV( 1)= 1    RQ= 1
 1

ORDER OF MATRIX:  0
```

Pascal

Listing

```
Program dr13(input,output);
{dr13.pas == run Nash svd for eigenvalue computations (Alg13)

          Copyright 1988 J.C.Nash
}
{constype.def ==
  This file contains various definitions and type statements which are
  used throughout the collection of "Compact Numerical Methods".  In many
  cases not all definitions are needed, and users with very tight memory
  constraints may wish to remove some of the lines of this file when
  compiling certain programs.

  Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}

const
  big = 1.0E+35;    {a very large number}
  Maxconst = 25;    {Maximum number of constants in data record}
  Maxobs = 100;     {Maximum number of observations in data record}
  Maxparm  = 25;    {Maximum number of parameters to adjust}
  Maxvars = 10;     {Maximum number of variables in data record}
  acctol = 0.0001;  {acceptable point tolerance for minimisation codes}
  maxm = 20;        {Maximum number or rows in a matrix}
  maxn = 20;        {Maximum number of columns in a matrix}
  maxmn = 40;       {maxn+maxm, the number of rows in a working array}
  maxsym = 210;     {maximum number of elements of a symmetric matrix
              which need to be stored = maxm * (maxm + 1)/2 }
  reltest = 10.0;   {a relative size used to check equality of numbers.
              Numbers x and y are considered equal if the
              floating-point representation of reltest+x equals
```

```
            that of reltest+y.}
  stepredn = 0.2;   {factor to reduce stepsize in line search}
  yearwrit = 1990;  {year in which file was written}

type
  str2 = string[2];
  rmatrix = array[1..maxm, 1..maxn] of real; {a real matrix}
  wmatrix = array[1..maxmn, 1..maxn] of real; {a working array, formed
                  as one real matrix stacked on another}
  smatvec = array[1..maxsym] of real; {a vector to store a symmetric matrix
            as the row-wise expansion of its lower triangle}
  rvector = array[1..maxm] of real;  {a real vector. We will use vectors
            of m elements always. While this is NOT space efficient,
            it simplifies program codes.}
  cgmethodtype= (Fletcher_Reeves,Polak_Ribiere,Beale_Sorenson);
    {three possible forms of the conjugate gradients updating formulae}
  probdata = record
          m     : integer; {number of observations}
          nvar  : integer; {number of variables}
          nconst: integer; {number of constants}
          vconst: array[1..Maxconst] of real;
          Ydata : array[1..Maxobs, 1..Maxvars] of real;
          nlls  : boolean; {true if problem is nonlinear least squares}
        end;
{
  NOTE: Pascal does not let us define the work-space for the function
  within the user-defined code.  This is a weakness of Pascal for this
  type of work.

}
var {global definitions}
  banner     : string[80]; {program name and description}

function calceps:real;
{calceps.pas ==
  This function returns the machine EPSILON or floating point tolerance,
  the smallest positive real number such that 1.0 + EPSILON > 1.0.
  EPSILON is needed to set various tolerances for different algorithms.
  While it could be entered as a constant, I prefer to calculate it, since
  users tend to move software between machines without paying attention to
  the computing environment. Note that more complete routines exist.
}
var
  e,e0: real;
  i: integer;
begin {calculate machine epsilon}
  e0 := 1; i:=0;
  repeat
    e0 := e0/2; e := 1+e0;  i := i+1;
  until (e=1.0) or (i=50); {note safety check}
  e0 := e0*2;
{ Writeln('Machine EPSILON =',e0);}
  calceps:=e0;
```

```
end; {calceps}

function resids(nRow, nCol: integer; A : rmatrix;
          Y: rvector; Bvec : rvector):real;
{resids.pas
  == Computes residuals and , if print is TRUE, displays them 7
    per line for the linear least squares problem. The sum of
    squared residuals is returned.

    residual vector = A * Bvec - Y
}
var
i, j: integer;
t1, ss : real;

begin
  writeln('Residuals');
  ss:=0.0;
  for i:=1 to nRow do
  begin
    t1:=-Y[i]; {note form of residual is residual = A * B - Y }
    for j:=1 to nCol do
      t1:=t1+A[i,j]*Bvec[j];
    ss:=ss+t1*t1;
    write(t1:10,' ');
    if (i = 7 * (i div 7)) and (i<nRow) then writeln;
  end; {loop on i}
  writeln;
  writeln('Sum of squared residuals =',ss);
  resids:=ss
end; {resids.pas == residual calculation for linear least squares}

procedure NashSVD(nRow, nCol: integer;
               var W: wmatrix;
               var Z: rvector);

var
  i, j, k, EstColRank, RotCount, SweepCount, slimit : integer;
  eps, e2, tol, vt, p, x0, y0, q, r, c0, s0, d1, d2 : real;

procedure rotate;
var
  ii : integer;

begin
  for ii := 1 to nRow+nCol do
  begin
    D1 := W[ii,j]; D2 := W[ii,k];
    W[ii,j] := D1*c0+D2*s0; W[ii,k] := -D1*s0+D2*c0
  end;
end;

begin
```

```
writeln('alg01.pas -- NashSVD');
eps := Calceps;
slimit := nCol div 4;  if slimit<6 then slimit := 6;

SweepCount := 0;
e2 := 10.0*nRow*eps*eps;
tol := eps*0.1;

EstColRank := nCol; ;

for i := 1 to nCol do
  begin
  for j := 1 to nCol do
    W[nRow+i,j] := 0.0;
  W[nRow+i,i] := 1.0;
end;

repeat
  RotCount := EstColRank*(EstColRank-1) div 2;
  SweepCount := SweepCount+1;

  for j := 1 to EstColRank-1 do
  begin
    for k := j+1 to EstColRank do
    begin
      p := 0.0;  q := 0.0; r := 0.0;
      for i := 1 to nRow do
      begin
        x0 := W[i,j]; y0 := W[i,k];
        p := p+x0*y0; q := q+x0*x0;  r := r+y0*y0;
      end;
      Z[j] := q; Z[k] := r;

      if q >= r then
      begin
        if (q<=e2*Z[1]) or (abs(p)<= tol*q) then RotCount := RotCount-1

        else
        begin
          p := p/q; r := 1-r/q; vt := sqrt(4*p*p + r*r);
          c0 := sqrt(0.5*(1+r/vt)); s0 := p/(vt*c0);
          rotate;
        end
      end
      else
      begin

        p := p/r; q := q/r-1; vt := sqrt(4*p*p + q*q);
        s0 := sqrt(0.5*(1-q/vt));
        if p<0 then s0 := -s0;
        c0 := p/(vt*s0);
        rotate;
      end;
```

```
      end;
    end;
    writeln('End of Sweep #', SweepCount,
            '-  no. of rotations performed =', RotCount);
    while (EstColRank >= 3) and (Z[EstColRank] <= Z[1]*tol + tol*tol)
          do EstColRank := EstColRank-1;
  until (RotCount=0) or (SweepCount>slimit);
  if (SweepCount > slimit) then writeln('**** SWEEP LIMIT EXCEEDED');
end;

Procedure evsvd(n: integer; var A,V : rmatrix; initev: boolean;
             W : wmatrix; var Z: rvector);

var
  i, j: integer;
  shift, t : real ;

begin
  writeln('alg13.pas -- symmetric matrix eigensolutions via svd');
  shift:=0.0;
  for i:=1 to n do
  begin
    t:=A[i,i];
    for j:=1 to n do
      if i<>j then t:=t-abs(A[i,j]);
    if t<shift then shift:=t;
  end;
  shift:=-shift;
  if shift<0.0 then shift:=0.0;
  writeln('Adding a shift of ',shift,' to diagonal of matrix.');
  for i:=1 to n do
  begin
    for j:=1 to n do
    begin
      W[i,j]:=A[i,j];
      if i=j then W[i,i]:=A[i,i]+shift;
      if initev then
      begin
        if i=j then W[i+n,i]:=0.0
        else
        begin
          W[i+n,j]:=0.0;
        end;
      end;
    end;
  end;
  if (n > 1) then
     NashSVD(n, n, W, Z)
  else
  begin { order 1 matrix }
     Z[1] := A[1,1]*A[1,1];
     W[2,1]:= 1.0; {Eigenvector!}
  end;
```

```
  for i:=1 to n do
  begin
    Z[i]:=sqrt(Z[i])-shift;
    for j:=1 to n do
      V[i,j]:=W[n+i,j];
  end;
end;

Procedure Frank2(var m, n: integer; var A: rmatrix);
var
  i,j: integer;
begin
    for i:=1 to m do
    begin
        for j:=1 to n do
        begin
          if (i <= j) then
             A[i,j]:=i
          else
             A[i,j]:=j;
        end;
    end;
end;

var
  i, j, nRow, nCol : integer;
  A, V, ACOPY : rmatrix;
  Bvec, Y, Z : rvector;
  W : wmatrix; {to store the working array}
  t1: real;
  initev : boolean;

begin
  banner:='dr13.pas -- driver for svd eigensolutions of a symmetric matrix';
  nRow := 1; {To get loop going}
  while (nRow > 0) do
  begin
  write('Order of problem (n): '); readln(nRow);
  if (nRow <= 0) then halt;
  nCol := nRow;
  Frank2(nRow, nCol, A);
  writeln('Initial matrix of order ', nRow);
  for j := 1 to nRow do
  begin
    for i := 1 to nRow do
    begin
      write(A[i,j]:10:5,' ');
      ACOPY[i,j] := A[i,j];
      W[i,j]:=0.0; {to avoid warning 'uninitialized' from fpc}
      if (7 * (i div 7) = i) and (i<nRow) then
      begin
        writeln;
      end;
```

```
    end;
    writeln;
  end;
  initev := true; {Here we want to get the eigenvectors of A, not some
             generalized problem.}
  writeln('Calling evsvd');
  evsvd( nRow, A, V, initev, W, Z);
  for j := 1 to nRow do
  begin
    t1 := Z[j];
    writeln;
    writeln('Eigenvalue ',j,' = ',t1);
    for i := 1 to nRow do
    begin
      write(V[i,j]:10:7,' ');
      if (i = 7 * (i div 7)) and (i<nRow) then
      begin
        writeln;
      end;
      Bvec[i] := V[i,j]; {to initialize residual test}
      Y[i] := t1*Bvec[i];
    end;
    writeln;
    t1 := resids(nRow, nCol, ACOPY, Y, Bvec);
  end; {loop on solutions j}
  end; {main while loop}
end. {dr13.pas}
```

Example output

```
fpc ../Pascal2021/dr13.pas
# copy to run file
mv ../Pascal2021/dr13 ../Pascal2021/dr13.run
../Pascal2021/dr13.run <../Pascal2021/dr13p.in >../Pascal2021/dr13p.out
```

```
## Free Pascal Compiler version 3.0.4+dfsg-23 [2019/11/25] for x86_64
## Copyright (c) 1993-2017 by Florian Klaempfl and others
## Target OS: Linux for x86-64
## Compiling ../Pascal2021/dr13.pas
## Linking ../Pascal2021/dr13
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 326 lines compiled, 0.1 sec
```

```
Order of problem (n): Initial matrix of order 2
   1.00000    1.00000
   1.00000    2.00000
Calling evsvd
alg13.pas -- symmetric matrix eigensolutions via svd
Adding a shift of -0.0000000000000000E+000 to diagonal of matrix.
alg01.pas -- NashSVD
End of Sweep #1-  no. of rotations performed =1
End of Sweep #2-  no. of rotations performed =0

Eigenvalue 1 =  2.6180339887498945E+000
```

```
 0.5257311  0.8506508
Residuals
 1.11E-016  4.44E-016
Sum of squared residuals = 2.0954117794933126E-031

Eigenvalue 2 =  3.8196601125010510E-001
-0.8506508  0.5257311
Residuals
 0.00E+000  0.00E+000
Sum of squared residuals = 0.0000000000000000E+000
Order of problem (n): Initial matrix of order 4
   1.00000    1.00000    1.00000    1.00000
   1.00000    2.00000    2.00000    2.00000
   1.00000    2.00000    3.00000    3.00000
   1.00000    2.00000    3.00000    4.00000
Calling evsvd
alg13.pas -- symmetric matrix eigensolutions via svd
Adding a shift of  3.0000000000000000E+000 to diagonal of matrix.
alg01.pas -- NashSVD
End of Sweep #1-  no. of rotations performed =6
End of Sweep #2-  no. of rotations performed =6
End of Sweep #3-  no. of rotations performed =6
End of Sweep #4-  no. of rotations performed =4
End of Sweep #5-  no. of rotations performed =0

Eigenvalue 1 =  8.2908593693815913E+000
 0.2280134  0.4285251  0.5773503  0.6565385
Residuals
-1.22E-015  2.22E-016 -1.55E-015 -2.22E-015
Sum of squared residuals = 8.8870111353804611E-030

Eigenvalue 2 =  1.0000000000000009E+000
-0.5773503 -0.5773503  0.0000000  0.5773503
Residuals
 6.66E-016  4.44E-016 -2.22E-016 -4.44E-016
Sum of squared residuals = 8.8746851837363828E-031

Eigenvalue 3 =  4.2602204776046149E-001
 0.6565385 -0.2280134 -0.5773503  0.4285251
Residuals
 2.78E-016 -1.11E-016  2.22E-016  6.66E-016
Sum of squared residuals = 5.8240121518270012E-031

Eigenvalue 4 =  2.8311858285794766E-001
-0.4285251  0.6565385 -0.5773503  0.2280134
Residuals
-3.89E-016  8.88E-016 -1.11E-016  5.55E-016
Sum of squared residuals = 1.2603285556070071E-030
Order of problem (n): Initial matrix of order 10
   1.00000    1.00000    1.00000    1.00000    1.00000    1.00000    1.00000
   1.00000    1.00000    1.00000
   1.00000    2.00000    2.00000    2.00000    2.00000    2.00000    2.00000
   2.00000    2.00000    2.00000
```

```
   1.00000    2.00000    3.00000    3.00000    3.00000    3.00000    3.00000
   3.00000    3.00000    3.00000
   1.00000    2.00000    3.00000    4.00000    4.00000    4.00000    4.00000
   4.00000    4.00000    4.00000
   1.00000    2.00000    3.00000    4.00000    5.00000    5.00000    5.00000
   5.00000    5.00000    5.00000
   1.00000    2.00000    3.00000    4.00000    5.00000    6.00000    6.00000
   6.00000    6.00000    6.00000
   1.00000    2.00000    3.00000    4.00000    5.00000    6.00000    7.00000
   7.00000    7.00000    7.00000
   1.00000    2.00000    3.00000    4.00000    5.00000    6.00000    7.00000
   8.00000    8.00000    8.00000
   1.00000    2.00000    3.00000    4.00000    5.00000    6.00000    7.00000
   8.00000    9.00000    9.00000
   1.00000    2.00000    3.00000    4.00000    5.00000    6.00000    7.00000
   8.00000    9.00000   10.00000
Calling evsvd
alg13.pas -- symmetric matrix eigensolutions via svd
Adding a shift of  3.6000000000000000E+001 to diagonal of matrix.
alg01.pas -- NashSVD
End of Sweep #1-  no. of rotations performed =45
End of Sweep #2-  no. of rotations performed =45
End of Sweep #3-  no. of rotations performed =45
End of Sweep #4-  no. of rotations performed =45
End of Sweep #5-  no. of rotations performed =16
End of Sweep #6-  no. of rotations performed =0

Eigenvalue 1 =  4.4766068652715035E+001
 0.0650474  0.1286417  0.1893624  0.2458530  0.2968517  0.3412192  0.3779645
 0.4062666  0.4254934  0.4352154
Residuals
 4.61E-015  3.89E-015  1.11E-015  6.00E-015  9.77E-015  4.00E-015  8.88E-016
 1.33E-015 -7.99E-015  1.78E-015
Sum of squared residuals = 2.5454630888977219E-028

Eigenvalue 2 =  5.0489173395222977E+000
-0.1893624 -0.3412192 -0.4254934 -0.4254934 -0.3412192 -0.1893624 -0.0000000
 0.1893624  0.3412192  0.4254934
Residuals
-2.05E-015 -7.66E-015 -5.33E-015  1.55E-015 -4.44E-015 -3.55E-015 -8.88E-016
 2.22E-015  1.78E-015  1.78E-015
Sum of squared residuals = 1.3809071775652032E-028

Eigenvalue 3 =  1.8730230604248987E+000
 0.2968517  0.4352154  0.3412192  0.0650474 -0.2458530 -0.4254934 -0.3779645
-0.1286417  0.1893624  0.4062666
Residuals
 9.55E-015  7.55E-015  3.11E-015 -2.66E-015  0.00E+000 -5.33E-015 -9.33E-015
 8.88E-016 -8.88E-016 -8.88E-016
Sum of squared residuals = 2.8265872310200379E-028

Eigenvalue 4 =  9.9999999999999289E-001
-0.3779645 -0.3779645 -0.0000000  0.3779645  0.3779645  0.0000000 -0.3779645
```

```
-0.3779645 -0.0000000  0.3779645
Residuals
-6.61E-015 -4.88E-015 -2.44E-015  2.66E-015  1.78E-015  8.88E-016 -4.44E-015
 4.44E-016  8.88E-016 -1.33E-015
Sum of squared residuals = 1.0699234175851075E-028

Eigenvalue 5 =  6.4310413210777284E-001
 0.4254934  0.1893624 -0.3412192 -0.3412192  0.1893624  0.4254934 -0.0000000
-0.4254934 -0.1893624  0.3412192
Residuals
 8.55E-015  1.18E-014 -6.88E-015 -1.33E-015  3.11E-015  7.11E-015  4.44E-015
-7.99E-015 -8.88E-016  5.33E-015
Sum of squared residuals = 4.3368860859689532E-028

Eigenvalue 6 =  4.6523308780856354E-001
-0.4352154  0.0650474  0.4254934 -0.1286417 -0.4062666  0.1893624  0.3779645
-0.2458530 -0.3412192  0.2968517
Residuals
-2.89E-015 -2.89E-015  1.55E-015 -6.22E-015  1.11E-015  3.11E-015 -1.33E-015
-3.11E-015 -1.33E-015 -7.99E-015
Sum of squared residuals = 1.4574205223958193E-028

Eigenvalue 7 =  3.6620887461579343E-001
 0.4062666 -0.2968517 -0.1893624  0.4352154 -0.1286417 -0.3412192  0.3779645
 0.0650474 -0.4254934  0.2458530
Residuals
 3.69E-015 -2.83E-015 -1.08E-014  3.33E-016 -4.22E-015 -6.88E-015 -1.78E-015
-5.11E-015 -2.22E-015 -6.66E-015
Sum of squared residuals = 2.8144846872495085E-028

Eigenvalue 8 =  3.0797852836987971E-001
-0.3412192  0.4254934 -0.1893624 -0.1893624  0.4254934 -0.3412192  0.0000000
 0.3412192 -0.4254934  0.1893624
Residuals
-5.61E-015  1.24E-014 -1.44E-015 -3.22E-015  1.39E-014 -3.77E-015 -1.33E-015
 1.24E-014 -7.77E-015  7.77E-015
Sum of squared residuals = 6.8252800187545925E-028

Eigenvalue 9 =  2.7378676163923643E-001
 0.2458530 -0.4062666  0.4254934 -0.2968517  0.0650474  0.1893624 -0.3779645
 0.4352154 -0.3412192  0.1286417
Residuals
-5.30E-015 -5.55E-017  7.33E-015 -3.66E-015 -2.33E-015 -3.33E-015 -8.44E-015
 6.00E-015 -1.78E-015 -1.11E-015
Sum of squared residuals = 2.2327613994072327E-028

Eigenvalue 10 =  2.5567956279643766E-001
-0.1286417  0.2458530 -0.3412192  0.4062666 -0.4352154  0.4254934 -0.3779645
 0.2968517 -0.1893624  0.0650474
Residuals
 1.18E-015 -2.25E-015 -1.67E-015 -5.00E-016  4.44E-015 -4.44E-016 -6.11E-016
 9.99E-016  9.99E-016  2.66E-015
Sum of squared residuals = 3.8856984778973554E-029
```

```
Order of problem (n): Initial matrix of order 1
   1.00000
Calling evsvd
alg13.pas -- symmetric matrix eigensolutions via svd
Adding a shift of -0.0000000000000000E+000 to diagonal of matrix.

Eigenvalue 1 =  1.0000000000000000E+000
 1.0000000
Residuals
 0.00E+000
Sum of squared residuals = 0.0000000000000000E+000
Order of problem (n):
```

Algorithm 14 – Jacobi symmetric matrix eigensolutions

Fortran

Pascal

Listing

```
Program dr14(input, output);
{dr14.pas == driver for Jacobi method (Alg14) for eigensolutions of a real
symmetric matrix
Copyright 1988 J.C.Nash
}
{constype.def ==
  This file contains various definitions and type statements which are
  used throughout the collection of "Compact Numerical Methods".  In many
  cases not all definitions are needed, and users with very tight memory
  constraints may wish to remove some of the lines of this file when
  compiling certain programs.

  Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}

const
  big = 1.0E+35;    {a very large number}
  Maxconst = 25;    {Maximum number of constants in data record}
  Maxobs = 100;     {Maximum number of observations in data record}
  Maxparm  = 25;    {Maximum number of parameters to adjust}
  Maxvars = 10;     {Maximum number of variables in data record}
  acctol = 0.0001;  {acceptable point tolerance for minimisation codes}
  maxm = 20;        {Maximum number or rows in a matrix}
  maxn = 20;        {Maximum number of columns in a matrix}
  maxmn = 40;       {maxn+maxm, the number of rows in a working array}
  maxsym = 210;     {maximum number of elements of a symmetric matrix
              which need to be stored = maxm * (maxm + 1)/2 }
  reltest = 10.0;   {a relative size used to check equality of numbers.
              Numbers x and y are considered equal if the
              floating-point representation of reltest+x equals
              that of reltest+y.}
  stepredn = 0.2;   {factor to reduce stepsize in line search}
  yearwrit = 1990;  {year in which file was written}

type
  str2 = string[2];
  rmatrix = array[1..maxm, 1..maxn] of real; {a real matrix}
  wmatrix = array[1..maxmn, 1..maxn] of real; {a working array, formed
                  as one real matrix stacked on another}
  smatvec = array[1..maxsym] of real; {a vector to store a symmetric matrix
              as the row-wise expansion of its lower triangle}
  rvector = array[1..maxm] of real;  {a real vector. We will use vectors
              of m elements always. While this is NOT space efficient,
              it simplifies program codes.}
  cgmethodtype= (Fletcher_Reeves,Polak_Ribiere,Beale_Sorenson);
```

```
    {three possible forms of the conjugate gradients updating formulae}
  probdata = record
          m     : integer; {number of observations}
          nvar  : integer; {number of variables}
          nconst: integer; {number of constants}
          vconst: array[1..Maxconst] of real;
          Ydata : array[1..Maxobs, 1..Maxvars] of real;
          nlls  : boolean; {true if problem is nonlinear least squares}
        end;
{
  NOTE: Pascal does not let us define the work-space for the function
  within the user-defined code.  This is a weakness of Pascal for this
  type of work.
}
var {global definitions}
  banner     : string[80]; {program name and description}


Procedure matrixin(var m, n: integer; var A: rmatrix;
              var avector: smatvec; var sym :boolean);

{matrixin.pas --

  This procedure generates an m by n real matrix in both
  A or avector.

  A is of type rmatrix, an array[1..nmax, 1..nmax] of real
  where nmax >= n for all possible n to be provided.

  avector is of type rvector, an array[1..nmax*(nmax+1)/2]
  of real, with nmax as above.

  sym is set true if the resulting matrix is symmetric.
}

var
  temp : real;
  i,j,k: integer;
  inchar: char;
  mtype: integer;
  mn : integer;

begin
  if (m=0) or (n=0) then
  begin
    writeln;
    writeln('******* Matrix dimensions zero *********');
    halt;
  end;
  writeln('Matrixin.pas -- generate or input a real matrix ',m,' by ',n);
  writeln('Possible matrices to generate:');
  writeln('0) Keyboard or console file input');
  writeln('1) Hilbert segment');
```

```
  writeln('2) Ding Dong');
  writeln('3) Moler');
  writeln('4) Frank symmetric');
  writeln('5) Bordered symmetric');
  writeln('6) Diagonal');
  writeln('7) Wilkinson W+');
  writeln('8) Wilkinson W-');
  writeln('9) Constant');
  writeln('10) Unit');
{ Note: others could be added.}
  mn:=n;
  if m>mn then mn:=m; {mn is maximum of m and n}
  write('Enter type to generate ');
  readln(mtype);
  writeln(mtype);
  case mtype of
    0: begin
      sym:=false;
      if m=n then
      begin
        write('Is matrix symmetric? ');  readln(inchar);
        writeln(inchar);
        if (inchar='y') or (inchar='Y') then sym:=true else sym:=false;
      end; {ask if symmetric}
      if sym then
      begin
        for i:=1 to n do
        begin
          writeln('Row ',i,' lower triangle elements');
          for j:=1 to i do
          begin
            read(A[i,j]);
            write(A[i,j]:10:5,' ');
            A[j,i]:=A[i,j];
            if (7*(j div 7) = j) and (j<i) then writeln;
          end;
          writeln;
        end;
      end {symmetric matrix}
      else
      begin {not symmetric}
        for i:=1 to m do
        begin
          writeln('Row ',i);
          for j:=1 to n do
          begin
            read(A[i,j]);
            write(A[i,j]:10:5,' ');
          end; {loop on j}
          writeln;
        end; {loop on i}
      end; {else not symmetric}
    end; {case 0 -- input of matrix}
```

```
1: begin {Hilbert}
  for i:=1 to mn do
    for j:=1 to mn do
      A[i,j]:=1.0/(i+j-1.0);
  if m=n then sym:=true;
end;
2: begin {Ding Dong}
  for i:=1 to mn do
    for j:=1 to mn do
      A[i,j]:=0.5/(1.5+n-i-j);
  if m=n then sym:=true;
end;
3: begin {Moler}
  for i:=1 to mn do
  begin
    for j:=1 to i do
    begin
      A[i,j]:=j-2.0;
      A[j,i]:=j-2.0;
    end;
    A[i,i]:=i;
    if m=n then sym:=true;
  end;
end;
4: begin {Frank symmetric}
  for i:=1 to mn do
    for j:=1 to i do
    begin
      A[i,j]:=j;
      A[j,i]:=j;
    end;
    if m=n then sym:=true;
end;
5: begin {Bordered}
  temp:=2.0;
  for i:=1 to (mn-1) do
  begin
    temp:=temp/2.0; {2^(1-i)}
    for j:=1 to mn do
      A[i,j]:=0.0;
    A[i,mn]:=temp;
    A[mn,i]:=temp;
    A[i,i]:=1.0;
  end;
  A[mn,mn]:=1.0;
  if m=n then sym:=true;
end;
6: begin {Diagonal}
  for i:=1 to mn do
  begin
    for j:=1 to mn do
      A[i,j]:=0.0;
    A[i,i]:=i;
```

```
    end;
    if m=n then sym:=true;
  end;
  7: begin {W+}
    k:=mn div 2; {[n/2]}
    for i:=1 to mn do
      for j:=1 to mn do
        A[i,j]:=0.0;
    if m=n then sym:=true;
    for i:=1 to k do
    begin
      A[i,i]:=k+1-i;
      A[mn-i+1,mn-i+1]:=k+1-i;
    end;
    for i:=1 to mn-1 do
    begin
      A[i,i+1]:=1.0;
      A[i+1,i]:=1.0;
    end;
  end;
  8: begin {W-}
    k:=mn div 2; {[n/2]}
    for i:=1 to mn do
      for j:=1 to mn do
        A[i,j]:=0.0;
    if m=n then sym:=true;
    for i:=1 to k do
    begin
      A[i,i]:=k+1-i;
      A[mn-i+1,mn-i+1]:=i-1-k;
    end;
    for i:=1 to mn-1 do
    begin
      A[i,i+1]:=1.0;
      A[i+1,i]:=1.0;
    end;
    if m=n then sym:=true;
  end;
  9: begin {Constant}
    write('Set all elements to a constant value = ');
    readln(temp);
    writeln(temp);
    for i:=1 to mn do
      for j:=1 to mn do
        A[i,j]:=temp;
    if m=n then sym:=true;
  end;
  10: begin {Unit}
    for i:=1 to mn do
    begin
      for j:=1 to mn do A[i,j]:=0.0;
      A[i,i]:=1.0;
    end;
```

```
      if m=n then sym:=true;
    end;
    else {case statement else} {!!!! Note missing close bracket here}
    begin
      writeln;
      writeln('*** ERROR *** unrecognized option');
      halt;
    end; {else of case statement}
  end; {case statement}
  if sym then
  begin {convert to vector form}
    k:=0; {index for vector element}
    for i:=1 to n do
    begin
      for j:=1 to i do
      begin
        k:=k+1;
        avector[k]:=A[i,j];
      end;
    end;
  end;
end; {matrixin}

function resids(nRow, nCol: integer; A : rmatrix;
          Y: rvector; Bvec : rvector; print : boolean):real;
{resids.pas
  == Computes residuals and , if print is TRUE, displays them 7
    per line for the linear least squares problem. The sum of
    squared residuals is returned.

    residual vector = A * Bvec - Y
}
var
i, j: integer;
t1, ss : real;

begin
  if print then
  begin
    writeln('Residuals');
  end;
  ss:=0.0;
  for i:=1 to nRow do
  begin
    t1:=-Y[i]; {note form of residual is residual = A * B - Y }
    for j:=1 to nCol do
      t1:=t1+A[i,j]*Bvec[j];
    ss:=ss+t1*t1;
    if print then
    begin
      write(t1:10,' ');
      if (i = 7 * (i div 7)) and (i<nRow) then writeln;
    end;
```

```
  end; {loop on i}
  if print then
  begin
    writeln;
    writeln('Sum of squared residuals =',ss);
  end;
  resids:=ss
end; {resids.pas == residual calculation for linear least squares}
```

```
Procedure evJacobi(n: integer;
               var A,V : rmatrix;
                   initev: boolean);


var
  count, i, j, k, limit, skipped : integer;
  c, p, q, s, t : real;
  oki, okj, rotn : boolean;

begin
  writeln('alg14.pas -- eigensolutions of a real symmetric');
  writeln('             matrix via a Jacobi method');
  if initev then
  begin
    for i := 1 to n do
    begin
      for j := 1 to n do V[i,j] := 0.0;
      V[i,i] := 1.0;
    end;
  end;
  count := 0;
  limit := 30;
  skipped := 0;

  while (count<=limit) and (skipped<((n*(n-1)) div 2) ) do

  begin
    count := count+1;
    write('sweep ',count,' ');
    skipped := 0;
    for i := 1 to (n-1) do
    begin
      for j := (i+1) to n do
      begin
        rotn := true;
        p := 0.5*(A[i,j]+A[j,i]);
        q := A[i,i]-A[j,j];
        t := sqrt(4.0*p*p+q*q);
```

```
        if t=0.0 then
        begin
          rotn := false;
        end
        else
        begin
          if q>=0.0 then
          begin

            oki := (abs(A[i,i])=abs(A[i,i])+100.0*abs(p));
            okj := (abs(A[j,j])=abs(A[j,j])+100.0*abs(p));
            if oki and okj then rotn := false
                           else rotn := true;

            if rotn then
            begin
               c := sqrt((t+q)/(2.0*t)); s := p/(t*c);
            end;
          end
          else
          begin
            rotn := true;
            s := sqrt((t-q)/(2.0*t));
            if p<0.0 then s := -s;
            c := p/(t*s);
          end;
          if 1.0=(1.0+abs(s)) then rotn := false;
        end;
        if rotn then
        begin
          for k := 1 to n do
          begin
            q := A[i,k]; A[i,k] := c*q+s*A[j,k]; A[j,k] := -s*q+c*A[j,k];
          end;

          for k := 1 to n do
          begin
            q := A[k,i]; A[k,i] := c*q+s*A[k,j]; A[k,j] := -s*q+c*A[k,j];

            q := V[k,i]; V[k,i] := c*q+s*V[k,j]; V[k,j] := -s*q+c*V[k,j];
          end;
        end
        else

           skipped := skipped+1;
      end;
    end;
    writeln('  ',skipped,' / ',n*(n-1) div 2,'  rotations skipped');
  end;
end;
{main program}
var
i, j, nRow, nCol : integer;
```

```
A, V, ACOPY : rmatrix;
Bvec, Y : rvector; {to test residuals}
t1: real;
tvec : smatvec; {needed only for Matrixin}
initev, sym : boolean;
begin
banner:='dr14.pas -- driver for Jacobi eigensolution method';

nRow := 1; {To get loop started}

while (nRow > 0) do
begin
  write('Order of problem (n) = ');
  readln(nRow);
  writeln(nRow);
  if (nRow <= 0) then halt;
  nCol:=nRow;
  Matrixin(nRow, nCol, A, tvec, sym);
  if not sym then halt; {program only designed for symmetric matrices}
  for j:=1 to nRow do
  begin
    for i:=1 to nRow do
    begin
      write(A[i, j]:10:5, ' ');
      ACOPY[i, j]:=A[i, j];
      if (7 * (i div 7) = i) and (i<nRow) then writeln;
    end;
    writeln;
  end;
  initev:=true; {Here we want to get the eigenvectors of A, not some
  generalized problem.}
  evJacobi( nRow, A, V, initev);
  for j:=1 to nRow do
  begin
    t1:=A[j, j];
    writeln('Eigenvalue ', j, ' = ', t1);
    for i:=1 to nRow do
    begin
      write(V[i, j]:10:7, ' ');
      if (i = 7 * (i div 7)) and (i<nRow) then writeln;
      Bvec[i]:=V[i, j]; {to initialize residual test}
      Y[i]:=t1*Bvec[i];
    end;
    writeln;
    t1 := resids(nRow, nCol, ACOPY, Y, Bvec, true);
    writeln;
  end; {loop on solutions j}
end; {while}
end. {dr14.pas}
```

Example output

```
fpc ../Pascal2021/dr14x.pas
# copy to run file
mv ../Pascal2021/dr14x ../Pascal2021/dr14x.run
../Pascal2021/dr14x.run <../Pascal2021/dr14xp.in >../Pascal2021/dr14xp.out
```

```
## Free Pascal Compiler version 3.0.4+dfsg-23 [2019/11/25] for x86_64
## Copyright (c) 1993-2017 by Florian Klaempfl and others
## Target OS: Linux for x86-64
## Compiling ../Pascal2021/dr14x.pas
## Linking ../Pascal2021/dr14x
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 467 lines compiled, 0.1 sec
```

```
Order of problem (n) = 5
Matrixin.pas -- generate or input a real matrix 5 by 5
Possible matrices to generate:
0) Keyboard or console file input
1) Hilbert segment
2) Ding Dong
3) Moler
4) Frank symmetric
5) Bordered symmetric
6) Diagonal
7) Wilkinson W+
8) Wilkinson W-
9) Constant
10) Unit
Enter type to generate 3
   1.00000   -1.00000   -1.00000   -1.00000   -1.00000
  -1.00000    2.00000    0.00000    0.00000    0.00000
  -1.00000    0.00000    3.00000    1.00000    1.00000
  -1.00000    0.00000    1.00000    4.00000    2.00000
  -1.00000    0.00000    1.00000    2.00000    5.00000
alg14.pas -- eigensolutions of a real symmetric
             matrix via a Jacobi method
sweep 1   0 / 10  rotations skipped
sweep 2   0 / 10  rotations skipped
sweep 3   0 / 10  rotations skipped
sweep 4   0 / 10  rotations skipped
sweep 5   7 / 10  rotations skipped
sweep 6   10 / 10  rotations skipped
Eigenvalue 1 =  7.4874999307049812E+000
 0.2556536 -0.0465883 -0.3403404 -0.5720713 -0.6995524
Residuals
 2.00E-015 -2.08E-016 -1.11E-015 -3.77E-015 -4.44E-016
Sum of squared residuals = 1.9715552247697695E-029

Eigenvalue 2 =  2.8289347824203293E+000
-0.3290435  0.3969474  0.6440739  0.1142105 -0.5534327
Residuals
-8.88E-016  1.33E-015  1.11E-015  8.88E-016 -4.44E-016
Sum of squared residuals = 4.7824692379023841E-030
```

```
Eigenvalue 3 =  2.3964685319861365E+000
-0.2547217  0.6424765 -0.0808234 -0.6002650  0.3943227
Residuals
-4.44E-016  6.66E-016  1.11E-016 -1.11E-015  0.00E+000
Sum of squared residuals = 1.8858706015439813E-030

Eigenvalue 4 =  2.2784504826045859E+000
-0.1364558  0.4900540 -0.6433686  0.5337402 -0.2059737
Residuals
-6.11E-016  1.11E-016  2.78E-016  9.99E-016 -2.22E-016
Sum of squared residuals = 1.5099290763995929E-030

Eigenvalue 5 =  8.6462722839592467E-003
-0.8618981 -0.4328202 -0.2210920 -0.1203898 -0.0801439
Residuals
-2.36E-016  2.22E-016  4.16E-017  2.50E-016  2.22E-016
Sum of squared residuals = 2.1840045569351255E-031

Order of problem (n) = 0
```

Algorithm 15 - Generalized symmetric eigenproblem

We aim to solve the generalized symmetric eigenproblem

$$Ax = eBx$$

for x and e, where symmetric matrices A and B and B is positive definite.

Fortran

Listing

```
C&&& A14-15
C  TEST ALG. 15   JULY 1978
C  J.C. NASH   JULY 1978, APRIL 1989
      LOGICAL FAIL
      INTEGER N,ND,I,NOUT,NIN
      REAL A(20,20),B(20,20),AT(20,20),Z(20),V(20,20),RMAX,VMAX
      EXTERNAL FRANKM,UNITM
      ND=20
C  I/O CHANNELS
      NIN=5
      NOUT=6
   1  READ(NIN,900)N
 900  FORMAT(I4)
      WRITE(NOUT,901)N
 901  FORMAT(' ORDER N=',I4)
      IF(N.LE.0)STOP
      ISWP=30
      FAIL=.FALSE.
      CALL UNITM(N,N,A,ND)
      CALL FRANKM(N,N,B,ND)
```

```
      CALL A15GSE(N,A,ND,B,ND,V,ND,FAIL,ISWP,NOUT)
      IF(FAIL)WRITE(NOUT,904)
 904  FORMAT(' FAILURE SET')
      DO 20 I=1,N
        Z(I)=A(I,I)
  20  CONTINUE
      CALL EVT(N,V,ND,Z,UNITM,FRANKM,AT,ND,B,ND,NOUT,RMAX,VMAX)
      GOTO 1
      END
      SUBROUTINE A14JE(N,A,NA,V,NV,ISWP,IPR,SETV,COMV)
C  ALGORITHM 14  JACOBI EIGENVALUES AND EIGENVECTORS OF REAL SYMMETRIC
C     MATRIX
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  N      = ORDER OF PROBLEM
C  A      = ARRAY CONTAINING MATRIX  -- MUST BE SYMMETRIC
C  NA     = FIRST DIMENSION OF A
C  V      = ARRAY INTO WHICH VECTORS COMPUTED
C  NV     = FIRST DIMENSION OF V
C  ISWP   =  SWEEP LIMIT (INPUT). SWEEP COUNT (OUTPUT)
C  IPR    = PRINT CHANNEL.  PRINTING ONLY IF IPR.GT.0
C  SETV   = LOGICAL SWITCH TO SET V INITIALLY TO IDENTITY OF ORDER N.
C  COMV   = LOGICAL SWITCH. IF .TRUE. THEN VECTORS TO BE CALCULATED.
C  STEP 0
      LOGICAL SETV,COMV
      INTEGER N,NA,NV,IPR,ISWP,LISWP,M,I,J,K,N1,I1
      REAL A(NA,N),V(NV,N),P,Q,T,S,C,FACT
C  FACTOR USED IN TEST AT STEP 7
      FACT=100.0
      N1=N-1
      LISWP=ISWP
      ISWP=0
C   EIGENVALUES LEFT IN DIAGONAL ELEMENTS OF A.
      IF(.NOT.COMV)GOTO 10
      IF(.NOT.SETV)GOTO 10
      DO 5 I=1,N
        DO 3 J=1,N
          V(I,J)=0.0
   3    CONTINUE
        V(I,I)=1.0
   5  CONTINUE
C  STEP 1
  10  ISWP=ISWP+1
      IF(ISWP.GT.LISWP)RETURN
      M=0
C  STEP 2
      IF(N.EQ.1)RETURN
      DO 160 I=1,N1
C  STEP 3
      I1=I+1
        DO 150 J=I1,N
C  STEP 4
          P=0.5*(A(I,J)+A(J,I))
          Q=A(I,I)-A(J,J)
```

```
          T=SQRT(4.0*P*P+Q*Q)
C   STEP 5
          IF(T.EQ.0.0)GOTO 110
C   STEP 6
          IF(Q.LT.0.0)GOTO 90
C   STEP 7
          IF(ABS(A(I,I)).LT.ABS(A(I,I))+FACT*ABS(P))GOTO 80
          IF(ABS(A(J,J)).EQ.ABS(A(J,J))+FACT*ABS(P))GOTO 110
C   STEP 8
  80      C=SQRT((T+Q)/(2.0*T))
          S=P/(T*C)
          GOTO 100
C   STEP 9
  90      S=SQRT((T-Q)/(2.0*T))
          IF(P.LT.0.0)S=-S
          C=P/(T*S)
C   STEP 10
 100      IF(1.0.LT.1.0+ABS(S))GOTO 120
C   STEP 11
 110      M=M+1
          GOTO 150
C   STEP 12
 120      DO 125 K=1,N
            Q=A(I,K)
            A(I,K)=C*Q+S*A(J,K)
            A(J,K)=-S*Q+C*A(J,K)
 125      CONTINUE
C   STEP 13
          DO 135 K=1,N
            Q=A(K,I)
            A(K,I)=C*Q+S*A(K,J)
            A(K,J)=-S*Q+C*A(K,J)
 135      CONTINUE
          IF(.NOT.COMV)GOTO 150
C   STEP 14
          DO 145 K=1,N
            Q=V(K,I)
            V(K,I)=C*Q+S*V(K,J)
            V(K,J)=-S*Q+C*V(K,J)
 145      CONTINUE
C   STEP 15
 150    CONTINUE
C   STEP 16
 160  CONTINUE
C   STEP 17
      IF(IPR.GT.0)WRITE(IPR,970)ISWP,M
 970  FORMAT( 9H AT SWEEP,I4,2X,I4,18H ROTATIONS SKIPPED)
      IF(M.LT.N*(N-1)/2)GOTO 10
      RETURN
      END
      SUBROUTINE A15GSE(N,A,NA,B,NB,V,NV,FAIL,ISWP,IPR)
C   ALGORITHM 15 GENERALISED SYMMETRIC EIGENPROBLEM BY 2 APPLICATIONS
C     OF JACOBI ALGORITHM 14
```

```
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  N     =   ORDER OF PROBLEM
C  A     =   A  MATRIX OF EIGENPROBLEM. DIAGONAL ELEMENTS BECOME
C  NA    =   FIRST DIMENSION OF A
C  B     =   B  MATRIX OF EIGENPROBLEM, MUST BE POSITIVE DEFINITE
C  NB    =   FIRST DIMENSION OF B
C  V     =   VECTOR MATRIX. ON OUTPUT COLUMNS ARE EIGENVECTORS
C            EIGENVALUES
C  FAIL  =   LOGICAL FLAG SET .TRUE. IF B NOT COMPUTATIONALLY
C              POSITIVE DEFINITE OR IF EITHER APPLICATION OF
C              ALGORITHM 14 TAKES MORE THAN ISWP SWEEPS
C  NV    =   FIRST DIMENSION OF V
C  ISWP  =   BOUND ON SWEEPS IN ALG. 14.
C  NA,NB,NV ALL .GE. N
C  IPR   =   PRINT CHANNEL.  IPR.GT.0 FOR PRINTING
C  STEP 0
      LOGICAL FAIL,COMV,SETV
      INTEGER N,NA,NB,NV,STAGE,ISWP,LISWP,I,J,K,M,IPR
      REAL A(NA,N),B(NB,N),V(NV,N),S
      FAIL=.FALSE.
      STAGE=1
      SETV=.TRUE.
      COMV=.TRUE.
C  STEP 1  INTERCHANGE - NOT GENERALLY EFFICIENT
 10   DO 16 I=1,N
        DO 14 J=1,N
          S=A(I,J)
          A(I,J)=B(I,J)
          B(I,J)=S
 14     CONTINUE
 16   CONTINUE
C  STEP 2
      LISWP=ISWP
      IF(IPR.GT.0)WRITE(IPR,964)STAGE
 964  FORMAT( 6H STAGE,I3)
      CALL A14JE(N,A,NA,V,NV,LISWP,IPR,SETV,COMV)
      IF(LISWP.GE.ISWP)FAIL=.TRUE.
      IF(FAIL)RETURN
C  STEP 3
      IF(STAGE.EQ.2)GOTO 80
C  STEP 4
      STAGE=2
      SETV=.FALSE.
      DO 46 I=1,N
        IF(A(I,I).LE.0.0)FAIL=.TRUE.
        IF(FAIL)RETURN
        S=1.0/SQRT(A(I,I))
        DO 44 J=1,N
          V(J,I)=S*V(J,I)
 44     CONTINUE
 46   CONTINUE
C  STEP 5
      DO 56 I=1,N
```

```
        DO 54 J=1,N
          A(I,J)=B(I,J)
  54    CONTINUE
  56  CONTINUE
C  STEP 6
      DO 68 I=1,N
        DO 66 J=I,N
          S=0.0
          DO 64 K=1,N
            DO 62 M=1,N
              S=S+V(K,I)*A(K,M)*V(M,J)
  62        CONTINUE
  64      CONTINUE
          B(I,J)=S
          B(J,I)=S
  66    CONTINUE
  68  CONTINUE
C  STEP 7
      GOTO 10
  80  ISWP=0
      RETURN
      END
      SUBROUTINE UNITM(M,N,A,NA)
C  PUTS UNIT MATRIX M BY N IN A
C  J.C. NASH   JULY 1978, APRIL 1989
      INTEGER M,N,NA,I,J
      REAL A(NA,N)
      DO 10 I=1,M
        DO 5 J=1,N
          A(I,J)=0.0
          IF(I.EQ.J)A(I,I)=1.0
   5    CONTINUE
  10  CONTINUE
      RETURN
      END
      SUBROUTINE EVT(N,V,NV,Z,AIN,BIN,A,NA,B,NB,NOUT,RMAX,VMAX)
C  J.C. NASH   JULY 1978, APRIL 1989
C  COMPUTES RESIDUALS AND INNER PRODUCTS
C    R  =  (A - Z(J)*B)*V(.,J)
C  AIN AND BIN ARE NAMES OF MATRIX CALCULATING ROUTINES FOR A AND B
C  WHOSE FIRST DIMENSIONS ARE NA AND NB RESP.
C  RMAX AND VMAX ARE MAX ABS RESIDUAL AND INNER PRODUCT RESP.
C
      INTEGER N,NV,NA,NB,NOUT,I,J,K,RPOSI,RPOSJ,VPOSI,VPOSJ,I1,N1
      REAL V(NV,N),A(NA,N),B(NB,N),Z(N),RMAX,VMAX
      DOUBLE PRECISION ACC,TACC,DABS,DBLE
      CALL AIN(N,N,A,NA)
      CALL BIN(N,N,B,NB)
      N1=N-1
      TACC=0.0
      RPOSI=1
      RPOSJ=1
      DO 20 I=1,N
```

```
        DO 15 J=1,N
          ACC=0.0
          DO 10 K=1,N
            ACC=ACC+DBLE(V(K,J))*(A(I,K)-Z(J)*B(I,K))
 10       CONTINUE
          IF(DABS(ACC).LE.TACC)GOTO 15
          TACC=DABS(ACC)
          RPOSI=I
          RPOSJ=J
 15     CONTINUE
 20   CONTINUE
      RMAX=TACC
      IF(NOUT.GT.0)WRITE(NOUT,951)RMAX,RPOSI,RPOSJ
 951  FORMAT(' MAX. ABS. RESIDUAL=',1PE16.8,'  POSN',2I4)
      VPOSI=0
      VPOSJ=0
      TACC=0.0
      IF(N.EQ.1)GOTO 45
      DO 40 I=1,N1
        I1=I+1
        DO 35 J=I1,N
          ACC=0.0
          DO 30 K=1,N
            ACC=ACC+DBLE(V(K,I))*V(K,J)
 30       CONTINUE
          IF(DABS(ACC).LE.TACC)GOTO 35
          TACC=DABS(ACC)
          VPOSI=I
          VPOSJ=J
 35     CONTINUE
 40   CONTINUE
      VMAX=TACC
      IF(NOUT.GT.0)WRITE(NOUT,952)VMAX,VPOSI,VPOSJ
 952  FORMAT(' MAX. ABS. INNER PRODUCT=',1PE16.8,'  POSN',2I4)
 45   IF(NOUT.LE.0)RETURN
      DO 50 J=1,N
        WRITE(NOUT,953)J,Z(J)
 953  FORMAT(' EIGENVALUE',I3,'=',1PE16.8)
        WRITE(NOUT,954)(V(K,J),K=1,N)
 954  FORMAT(1H ,5E16.8)
 50   CONTINUE
      RETURN
      END
      SUBROUTINE FRANKM(M,N,A,NA)
C  J.C. NASH   JULY 1978, APRIL 1989
      INTEGER M,N,NA,I,J
C  INPUTS FRANK MATRIX M BY N INTO A
      REAL A(NA,N)
      DO 20 I=1,M
        DO 10 J=1,N
          A(I,J)=AMIN0(I,J)
 10     CONTINUE
 20   CONTINUE
```

```
      RETURN
      END
```

Example output

```
## #!/bin/bash
gfortran ../fortran/dr1415.f
mv ./a.out ../fortran/dr1415.run
# use dr13 here as well as in Alg 13
../fortran/dr1415.run < ../fortran/dr13.in
```

```
##   ORDER N=    2
##   STAGE   1
##   AT SWEEP    1      0 ROTATIONS SKIPPED
##   AT SWEEP    2      1 ROTATIONS SKIPPED
##   STAGE   2
##   AT SWEEP    1      0 ROTATIONS SKIPPED
##   AT SWEEP    2      1 ROTATIONS SKIPPED
##   MAX. ABS. RESIDUAL=  2.51580275E-07   POSN    2    1
##   MAX. ABS. INNER PRODUCT=  7.73007969E-09   POSN    1    2
##   EIGENVALUE   1=  2.61803389E+00
##     0.13763819E+01 -0.85065085E+00
##   EIGENVALUE   2=  3.81965995E-01
##     0.32491970E+00  0.52573109E+00
##   ORDER N=    4
##   STAGE   1
##   AT SWEEP    1      0 ROTATIONS SKIPPED
##   AT SWEEP    2      0 ROTATIONS SKIPPED
##   AT SWEEP    3      1 ROTATIONS SKIPPED
##   AT SWEEP    4      6 ROTATIONS SKIPPED
##   STAGE   2
##   AT SWEEP    1      0 ROTATIONS SKIPPED
##   AT SWEEP    2      5 ROTATIONS SKIPPED
##   AT SWEEP    3      6 ROTATIONS SKIPPED
##   MAX. ABS. RESIDUAL=  9.88243642E-07   POSN    3    1
##   MAX. ABS. INNER PRODUCT=  5.05645765E-08   POSN    1    3
##   EIGENVALUE   1=  3.53208756E+00
##    -0.80536371E+00  0.12338886E+01 -0.10850633E+01  0.42852503E+00
##   EIGENVALUE   2=  2.34729719E+00
##    -0.10058755E+01  0.34933683E+00  0.88455212E+00 -0.65653861E+00
##   EIGENVALUE   3=  1.00000060E+00
##    -0.57735038E+00 -0.57735050E+00 -0.25529275E-07  0.57735038E+00
##   EIGENVALUE   4=  1.20614767E-01
##    -0.79188228E-01 -0.14882518E+00 -0.20051166E+00 -0.22801344E+00
##   ORDER N=    4
##   STAGE   1
##   AT SWEEP    1      0 ROTATIONS SKIPPED
##   AT SWEEP    2      0 ROTATIONS SKIPPED
##   AT SWEEP    3      1 ROTATIONS SKIPPED
##   AT SWEEP    4      6 ROTATIONS SKIPPED
##   STAGE   2
##   AT SWEEP    1      0 ROTATIONS SKIPPED
##   AT SWEEP    2      5 ROTATIONS SKIPPED
##   AT SWEEP    3      6 ROTATIONS SKIPPED
```

```
## MAX. ABS. RESIDUAL=  9.88243642E-07  POSN   3   1
## MAX. ABS. INNER PRODUCT=  5.05645765E-08  POSN   1   3
## EIGENVALUE  1=  3.53208756E+00
##   -0.80536371E+00  0.12338886E+01 -0.10850633E+01  0.42852503E+00
## EIGENVALUE  2=  2.34729719E+00
##   -0.10058755E+01  0.34933683E+00  0.88455212E+00 -0.65653861E+00
## EIGENVALUE  3=  1.00000060E+00
##   -0.57735038E+00 -0.57735050E+00 -0.25529275E-07  0.57735038E+00
## EIGENVALUE  4=  1.20614767E-01
##   -0.79188228E-01 -0.14882518E+00 -0.20051166E+00 -0.22801344E+00
## ORDER N=  10
## STAGE  1
## AT SWEEP   1     0 ROTATIONS SKIPPED
## AT SWEEP   2     0 ROTATIONS SKIPPED
## AT SWEEP   3     0 ROTATIONS SKIPPED
## AT SWEEP   4    23 ROTATIONS SKIPPED
## AT SWEEP   5    45 ROTATIONS SKIPPED
## STAGE  2
## AT SWEEP   1     0 ROTATIONS SKIPPED
## AT SWEEP   2    35 ROTATIONS SKIPPED
## AT SWEEP   3    45 ROTATIONS SKIPPED
## MAX. ABS. RESIDUAL=  7.95809774E-06  POSN  10   1
## MAX. ABS. INNER PRODUCT=  1.70029864E-07  POSN   1   3
## EIGENVALUE  1=  3.91114664E+00
##    0.25440717E+00 -0.48620966E+00  0.67481190E+00 -0.80345494E+00  0.86070871E+00
##   -0.84148449E+00  0.74749005E+00 -0.58707643E+00  0.37449750E+00 -0.12864262E+00
## EIGENVALUE  2=  3.65247846E+00
##   -0.46986169E+00  0.77643681E+00 -0.81318295E+00  0.56733066E+00 -0.12432010E+00
##   -0.36189401E+00  0.72234160E+00 -0.83175814E+00  0.65211862E+00 -0.24585268E+00
## EIGENVALUE  3=  3.24698114E+00
##   -0.61485553E+00  0.76671326E+00 -0.34122097E+00 -0.34121776E+00  0.76671290E+00
##   -0.61485648E+00  0.70596684E-09  0.61485595E+00 -0.76671231E+00  0.34121889E+00
## EIGENVALUE  4=  2.73068285E+00
##   -0.67134637E+00  0.49054128E+00  0.31291714E+00 -0.71918464E+00  0.21257788E+00
##    0.56385678E+00 -0.62457776E+00 -0.10748890E+00  0.70311815E+00 -0.40626636E+00
## EIGENVALUE  5=  2.14946055E+00
##    0.63807106E+00 -0.95366970E-01 -0.62381732E+00  0.18860267E+00  0.59562886E+00
##   -0.27762464E+00 -0.55413532E+00  0.36044526E+00  0.50026351E+00 -0.43521550E+00
## EIGENVALUE  6=  1.55495822E+00
##   -0.53058165E+00 -0.23613074E+00  0.42549348E+00  0.42549339E+00 -0.23613124E+00
##   -0.53058165E+00  0.14518602E-06  0.53058153E+00  0.23613124E+00 -0.42549354E+00
## EIGENVALUE  7=  1.00000012E+00
##   -0.37796488E+00 -0.37796432E+00  0.30240781E-06  0.37796441E+00  0.37796423E+00
##   -0.16945556E-08 -0.37796435E+00 -0.37796468E+00  0.11743472E-06  0.37796459E+00
## EIGENVALUE  8=  5.33896327E-01
##   -0.21690433E+00 -0.31800413E+00 -0.24932273E+00 -0.47528878E-01  0.17964047E+00
##    0.31090042E+00  0.27617189E+00  0.93996122E-01 -0.13836364E+00 -0.29685175E+00
## EIGENVALUE  9=  1.98062271E-01
##    0.84274180E-01  0.15185684E+00  0.18936239E+00  0.18936241E+00  0.15185687E+00
##    0.84274203E-01 -0.56293572E-07 -0.84274210E-01 -0.15185684E+00 -0.18936238E+00
## EIGENVALUE 10=  2.23383494E-02
##    0.97219953E-02  0.19226812E-01  0.28302142E-01  0.36745246E-01  0.44367522E-01
##    0.50998691E-01  0.56490630E-01  0.60720690E-01  0.63594334E-01  0.65047383E-01
## ORDER N=   1
```

```
## STAGE 1
## STAGE 2
## MAX. ABS. RESIDUAL= 0.00000000E+00 POSN 1 1
## EIGENVALUE 1= 1.00000000E+00
## 0.10000000E+01
## ORDER N= 0
## Note: The following floating-point exceptions are signalling: IEEE_UNDERFLOW_FLAG IEEE_DENORMAL
```

Pascal

Listing

```
Program dr14(input, output);
{dr14.pas == driver for Jacobi method (Alg14) for eigensolutions of a real
symmetric matrix
Copyright 1988 J.C.Nash
}
{constype.def ==
  This file contains various definitions and type statements which are
  used throughout the collection of "Compact Numerical Methods".  In many
  cases not all definitions are needed, and users with very tight memory
  constraints may wish to remove some of the lines of this file when
  compiling certain programs.

  Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}

const
  big = 1.0E+35;    {a very large number}
  Maxconst = 25;    {Maximum number of constants in data record}
  Maxobs = 100;     {Maximum number of observations in data record}
  Maxparm  = 25;    {Maximum number of parameters to adjust}
  Maxvars = 10;     {Maximum number of variables in data record}
  acctol = 0.0001;  {acceptable point tolerance for minimisation codes}
  maxm = 20;        {Maximum number or rows in a matrix}
  maxn = 20;        {Maximum number of columns in a matrix}
  maxmn = 40;       {maxn+maxm, the number of rows in a working array}
  maxsym = 210;     {maximum number of elements of a symmetric matrix
              which need to be stored = maxm * (maxm + 1)/2 }
  reltest = 10.0;   {a relative size used to check equality of numbers.
              Numbers x and y are considered equal if the
              floating-point representation of reltest+x equals
              that of reltest+y.}
  stepredn = 0.2;   {factor to reduce stepsize in line search}
  yearwrit = 1990;  {year in which file was written}

type
  str2 = string[2];
  rmatrix = array[1..maxm, 1..maxn] of real; {a real matrix}
  wmatrix = array[1..maxmn, 1..maxn] of real; {a working array, formed
                as one real matrix stacked on another}
  smatvec = array[1..maxsym] of real; {a vector to store a symmetric matrix
```

```
              as the row-wise expansion of its lower triangle}
  rvector = array[1..maxm] of real;  {a real vector. We will use vectors
              of m elements always. While this is NOT space efficient,
              it simplifies program codes.}
  cgmethodtype= (Fletcher_Reeves,Polak_Ribiere,Beale_Sorenson);
    {three possible forms of the conjugate gradients updating formulae}
  probdata = record
          m     : integer; {number of observations}
          nvar  : integer; {number of variables}
          nconst: integer; {number of constants}
          vconst: array[1..Maxconst] of real;
          Ydata : array[1..Maxobs, 1..Maxvars] of real;
          nlls  : boolean; {true if problem is nonlinear least squares}
        end;
{
  NOTE: Pascal does not let us define the work-space for the function
  within the user-defined code.  This is a weakness of Pascal for this
  type of work.
}
var {global definitions}
  banner     : string[80]; {program name and description}


Procedure matrixin(var m, n: integer; var A: rmatrix;
              var avector: smatvec; var sym :boolean);

{matrixin.pas --

  This procedure generates an m by n real matrix in both
  A or avector.

  A is of type rmatrix, an array[1..nmax, 1..nmax] of real
  where nmax >= n for all possible n to be provided.

  avector is of type rvector, an array[1..nmax*(nmax+1)/2]
  of real, with nmax as above.

  sym is set true if the resulting matrix is symmetric.
}

var
  temp : real;
  i,j,k: integer;
  inchar: char;
  mtype: integer;
  mn : integer;

begin
  if (m=0) or (n=0) then
  begin
    writeln;
    writeln('******* Matrix dimensions zero *********');
    halt;
```

```
  end;
  writeln('Matrixin.pas -- generate or input a real matrix ',m,' by ',n);
  writeln('Possible matrices to generate:');
  writeln('0) Keyboard or console file input');
  writeln('1) Hilbert segment');
  writeln('2) Ding Dong');
  writeln('3) Moler');
  writeln('4) Frank symmetric');
  writeln('5) Bordered symmetric');
  writeln('6) Diagonal');
  writeln('7) Wilkinson W+');
  writeln('8) Wilkinson W-');
  writeln('9) Constant');
  writeln('10) Unit');
{ Note: others could be added.}
  mn:=n;
  if m>mn then mn:=m; {mn is maximum of m and n}
  write('Enter type to generate ');
  readln(mtype);
  writeln(mtype);
  case mtype of
    0: begin
      sym:=false;
      if m=n then
      begin
        write('Is matrix symmetric? ');  readln(inchar);
        writeln(inchar);
        if (inchar='y') or (inchar='Y') then sym:=true else sym:=false;
      end; {ask if symmetric}
      if sym then
      begin
        for i:=1 to n do
        begin
          writeln('Row ',i,' lower triangle elements');
          for j:=1 to i do
          begin
            read(A[i,j]);
            write(A[i,j]:10:5,' ');
            A[j,i]:=A[i,j];
            if (7*(j div 7) = j) and (j<i) then writeln;
          end;
          writeln;
        end;
      end {symmetric matrix}
      else
      begin {not symmetric}
        for i:=1 to m do
        begin
          writeln('Row ',i);
          for j:=1 to n do
          begin
            read(A[i,j]);
            write(A[i,j]:10:5,' ');
```

```
        end; {loop on j}
        writeln;
      end; {loop on i}
    end; {else not symmetric}
  end; {case 0 -- input of matrix}
  1: begin {Hilbert}
    for i:=1 to mn do
      for j:=1 to mn do
        A[i,j]:=1.0/(i+j-1.0);
    if m=n then sym:=true;
  end;
  2: begin {Ding Dong}
    for i:=1 to mn do
      for j:=1 to mn do
        A[i,j]:=0.5/(1.5+n-i-j);
    if m=n then sym:=true;
  end;
  3: begin {Moler}
    for i:=1 to mn do
    begin
      for j:=1 to i do
      begin
        A[i,j]:=j-2.0;
        A[j,i]:=j-2.0;
      end;
      A[i,i]:=i;
      if m=n then sym:=true;
    end;
  end;
  4: begin {Frank symmetric}
    for i:=1 to mn do
      for j:=1 to i do
      begin
        A[i,j]:=j;
        A[j,i]:=j;
      end;
      if m=n then sym:=true;
  end;
  5: begin {Bordered}
    temp:=2.0;
    for i:=1 to (mn-1) do
    begin
      temp:=temp/2.0; {2^(1-i)}
      for j:=1 to mn do
        A[i,j]:=0.0;
      A[i,mn]:=temp;
      A[mn,i]:=temp;
      A[i,i]:=1.0;
    end;
    A[mn,mn]:=1.0;
    if m=n then sym:=true;
  end;
  6: begin {Diagonal}
```

```
    for i:=1 to mn do
    begin
      for j:=1 to mn do
        A[i,j]:=0.0;
      A[i,i]:=i;
    end;
    if m=n then sym:=true;
  end;
  7: begin {W+}
    k:=mn div 2; {[n/2]}
    for i:=1 to mn do
      for j:=1 to mn do
        A[i,j]:=0.0;
    if m=n then sym:=true;
    for i:=1 to k do
    begin
      A[i,i]:=k+1-i;
      A[mn-i+1,mn-i+1]:=k+1-i;
    end;
    for i:=1 to mn-1 do
    begin
      A[i,i+1]:=1.0;
      A[i+1,i]:=1.0;
    end;
  end;
  8: begin {W-}
    k:=mn div 2; {[n/2]}
    for i:=1 to mn do
      for j:=1 to mn do
        A[i,j]:=0.0;
    if m=n then sym:=true;
    for i:=1 to k do
    begin
      A[i,i]:=k+1-i;
      A[mn-i+1,mn-i+1]:=i-1-k;
    end;
    for i:=1 to mn-1 do
    begin
      A[i,i+1]:=1.0;
      A[i+1,i]:=1.0;
    end;
    if m=n then sym:=true;
  end;
  9: begin {Constant}
    write('Set all elements to a constant value = ');
    readln(temp);
    writeln(temp);
    for i:=1 to mn do
      for j:=1 to mn do
        A[i,j]:=temp;
    if m=n then sym:=true;
  end;
  10: begin {Unit}
```

```
      for i:=1 to mn do
      begin
        for j:=1 to mn do A[i,j]:=0.0;
        A[i,i]:=1.0;
      end;
      if m=n then sym:=true;
    end;
    else {case statement else} {!!!! Note missing close bracket here}
    begin
      writeln;
      writeln('*** ERROR *** unrecognized option');
      halt;
    end; {else of case statement}
  end; {case statement}
  if sym then
  begin {convert to vector form}
    k:=0; {index for vector element}
    for i:=1 to n do
    begin
      for j:=1 to i do
      begin
        k:=k+1;
        avector[k]:=A[i,j];
      end;
    end;
  end;
end; {matrixin}

function resids(nRow, nCol: integer; A : rmatrix;
          Y: rvector; Bvec : rvector; print : boolean):real;
{resids.pas
  == Computes residuals and , if print is TRUE, displays them 7
    per line for the linear least squares problem. The sum of
    squared residuals is returned.

    residual vector = A * Bvec - Y
}
var
i, j: integer;
t1, ss : real;

begin
  if print then
  begin
    writeln('Residuals');
  end;
  ss:=0.0;
  for i:=1 to nRow do
  begin
    t1:=-Y[i]; {note form of residual is residual = A * B - Y }
    for j:=1 to nCol do
      t1:=t1+A[i,j]*Bvec[j];
    ss:=ss+t1*t1;
```

```
    if print then
    begin
      write(t1:10,' ');
      if (i = 7 * (i div 7)) and (i<nRow) then writeln;
    end;
  end; {loop on i}
  if print then
  begin
    writeln;
    writeln('Sum of squared residuals =',ss);
  end;
  resids:=ss
end; {resids.pas == residual calculation for linear least squares}
```

```
Procedure evJacobi(n: integer;
               var A,V : rmatrix;
                   initev: boolean);


var
  count, i, j, k, limit, skipped : integer;
  c, p, q, s, t : real;
  oki, okj, rotn : boolean;

begin
  writeln('alg14.pas -- eigensolutions of a real symmetric');
  writeln('             matrix via a Jacobi method');
  if initev then
  begin
    for i := 1 to n do
    begin
      for j := 1 to n do V[i,j] := 0.0;
      V[i,i] := 1.0;
    end;
  end;
  count := 0;
  limit := 30;
  skipped := 0;

  while (count<=limit) and (skipped<((n*(n-1)) div 2) ) do

  begin
    count := count+1;
    write('sweep ',count,' ');
    skipped := 0;
    for i := 1 to (n-1) do
    begin
      for j := (i+1) to n do
```

```
    begin
      rotn := true;
      p := 0.5*(A[i,j]+A[j,i]);
      q := A[i,i]-A[j,j];
      t := sqrt(4.0*p*p+q*q);
      if t=0.0 then
      begin
        rotn := false;
      end
      else
      begin
        if q>=0.0 then
        begin

          oki := (abs(A[i,i])=abs(A[i,i])+100.0*abs(p));
          okj := (abs(A[j,j])=abs(A[j,j])+100.0*abs(p));
          if oki and okj then rotn := false
                         else rotn := true;

          if rotn then
          begin
             c := sqrt((t+q)/(2.0*t)); s := p/(t*c);
          end;
        end
        else
        begin
          rotn := true;
          s := sqrt((t-q)/(2.0*t));
          if p<0.0 then s := -s;
          c := p/(t*s);
        end;
        if 1.0=(1.0+abs(s)) then rotn := false;
      end;
      if rotn then
      begin
        for k := 1 to n do
        begin
          q := A[i,k]; A[i,k] := c*q+s*A[j,k]; A[j,k] := -s*q+c*A[j,k];
        end;

        for k := 1 to n do
        begin
          q := A[k,i]; A[k,i] := c*q+s*A[k,j]; A[k,j] := -s*q+c*A[k,j];

          q := V[k,i]; V[k,i] := c*q+s*V[k,j]; V[k,j] := -s*q+c*V[k,j];
        end;
      end
      else

         skipped := skipped+1;
    end;
  end;
  writeln('  ',skipped,' / ',n*(n-1) div 2,'  rotations skipped');
```

```
  end;
end;
{main program}
var
i, j, nRow, nCol : integer;
A, V, ACOPY : rmatrix;
Bvec, Y : rvector; {to test residuals}
t1: real;
tvec : smatvec; {needed only for Matrixin}
initev, sym : boolean;
begin
banner:='dr14.pas -- driver for Jacobi eigensolution method';

nRow := 1; {To get loop started}

while (nRow > 0) do
begin
  write('Order of problem (n) = ');
  readln(nRow);
  writeln(nRow);
  if (nRow <= 0) then halt;
  nCol:=nRow;
  Matrixin(nRow, nCol, A, tvec, sym);
  if not sym then halt; {program only designed for symmetric matrices}
  for j:=1 to nRow do
  begin
    for i:=1 to nRow do
    begin
      write(A[i, j]:10:5, ' ');
      ACOPY[i, j]:=A[i, j];
      if (7 * (i div 7) = i) and (i<nRow) then writeln;
    end;
    writeln;
  end;
  initev:=true; {Here we want to get the eigenvectors of A, not some
  generalized problem.}
  evJacobi( nRow, A, V, initev);
  for j:=1 to nRow do
  begin
    t1:=A[j, j];
    writeln('Eigenvalue ', j, ' = ', t1);
    for i:=1 to nRow do
    begin
      write(V[i, j]:10:7, ' ');
      if (i = 7 * (i div 7)) and (i<nRow) then writeln;
      Bvec[i]:=V[i, j]; {to initialize residual test}
      Y[i]:=t1*Bvec[i];
    end;
    writeln;
    t1 := resids(nRow, nCol, ACOPY, Y, Bvec, true);
    writeln;
  end; {loop on solutions j}
end; {while}
```

```
end. {dr14.pas}
```

Example output

```
fpc ../Pascal2021/dr15x.pas
# copy to run file
mv ../Pascal2021/dr15x ../Pascal2021/dr15x.run
../Pascal2021/dr15x.run <../Pascal2021/dr15xp.in >../Pascal2021/dr15xp.out
```

```
## Free Pascal Compiler version 3.0.4+dfsg-23 [2019/11/25] for x86_64
## Copyright (c) 1993-2017 by Florian Klaempfl and others
## Target OS: Linux for x86-64
## Compiling ../Pascal2021/dr15x.pas
## dr15x.pas(356,3) Note: Local variable "ch" not used
## dr15x.pas(477,7) Note: Local variable "k" not used
## dr15x.pas(478,1) Note: Local variable "s" not used
## dr15x.pas(478,4) Note: Local variable "s2" is assigned but never used
## dr15x.pas(480,4) Note: Local variable "Y" not used
## dr15x.pas(482,1) Note: Local variable "ch" not used
## Linking ../Pascal2021/dr15x
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 537 lines compiled, 0.1 sec
## 6 note(s) issued
```

```
Order of matrices = 5
Provide matrix A
Matrixin.pas -- generate or input a real matrix 5 by 5
Possible matrices to generate:
0) Keyboard or console file input
1) Hilbert segment
2) Ding Dong
3) Moler
4) Frank symmetric
5) Bordered symmetric
6) Diagonal
7) Wilkinson W+
8) Wilkinson W-
9) Constant
10) Unit
Enter type to generate 4
   1.00000    1.00000    1.00000    1.00000    1.00000
   1.00000    2.00000    2.00000    2.00000    2.00000
   1.00000    2.00000    3.00000    3.00000    3.00000
   1.00000    2.00000    3.00000    4.00000    4.00000
   1.00000    2.00000    3.00000    4.00000    5.00000
Provide matrix B
Matrixin.pas -- generate or input a real matrix 5 by 5
Possible matrices to generate:
0) Keyboard or console file input
1) Hilbert segment
2) Ding Dong
3) Moler
4) Frank symmetric
5) Bordered symmetric
```

```
6) Diagonal
7) Wilkinson W+
8) Wilkinson W-
9) Constant
10) Unit
Enter type to generate 3
   1.00000   -1.00000   -1.00000   -1.00000   -1.00000
  -1.00000    2.00000    0.00000    0.00000    0.00000
  -1.00000    0.00000    3.00000    1.00000    1.00000
  -1.00000    0.00000    1.00000    4.00000    2.00000
  -1.00000    0.00000    1.00000    2.00000    5.00000
alg15.pas -- generalized symmetric matrix eigenproblem
Eigensolutions of metric B
alg14.pas -- eigensolutions of a real symmetric
             matrix via a Jacobi method
sweep 1   0 / 10  rotations skipped
sweep 2   0 / 10  rotations skipped
sweep 3   0 / 10  rotations skipped
sweep 4   0 / 10  rotations skipped
sweep 5   7 / 10  rotations skipped
sweep 6   10 / 10  rotations skipped
Eigensolutions of general problem
alg14.pas -- eigensolutions of a real symmetric
             matrix via a Jacobi method
sweep 1   0 / 10  rotations skipped
sweep 2   0 / 10  rotations skipped
sweep 3   0 / 10  rotations skipped
sweep 4   3 / 10  rotations skipped
sweep 5   10 / 10  rotations skipped

Eigenvalue 1 =  4.5198745112453821E+002
-9.2524772 -4.6568542 -2.3845068 -1.3017749 -0.8684904
Generalized matrix eigensolution residuals
 3.98E-013 -7.11E-013  0.00E+000 -5.68E-013 -6.82E-013
Sum of squared residuals = 1.4525521166230544E-024
Rayleigh quotient =  4.5198745112453753E+002
Generalized matrix eigensolution residuals
 1.71E-013 -7.11E-013  3.98E-013 -4.55E-013 -2.27E-013
Sum of squared residuals = 9.5172010511633844E-025

Eigenvalue 2 =  4.9797005187099336E-001
 0.3224789  0.2680005  0.0784294 -0.1506762 -0.3038294
Generalized matrix eigensolution residuals
-2.22E-016  1.11E-016  2.22E-016  3.33E-016  5.55E-016
Sum of squared residuals = 5.3001592069536731E-031
Rayleigh quotient =  4.9797005187099297E-001
Generalized matrix eigensolution residuals
-1.11E-016  2.22E-016  2.22E-016  0.00E+000 -1.11E-016
Sum of squared residuals = 1.2325951644078309E-031

Eigenvalue 3 =  2.4095744686996778E-001
 0.3782595  0.0138457 -0.3723758 -0.1720849  0.2992493
Generalized matrix eigensolution residuals
```

```
 5.55E-017  6.66E-016  8.88E-016  8.88E-016  8.88E-016
Sum of squared residuals = 2.8133984627608741E-030
Rayleigh quotient =  2.4095744686996767E-001
Generalized matrix eigensolution residuals
 1.67E-016  5.55E-016  5.55E-016  8.88E-016  1.11E-015
Sum of squared residuals = 2.6654870430319344E-030

Eigenvalue 4 =  1.5349826485584656E-001
 0.3288484 -0.3365795 -0.0739346  0.3925750 -0.2233883
Generalized matrix eigensolution residuals
-1.94E-015 -2.89E-015 -3.33E-015 -3.55E-015 -3.66E-015
Sum of squared residuals = 4.9245258306003866E-029
Rayleigh quotient =  1.5349826485584656E-001
Generalized matrix eigensolution residuals
-1.94E-015 -2.89E-015 -3.33E-015 -3.55E-015 -3.66E-015
Sum of squared residuals = 4.9245258306003866E-029

Eigenvalue 5 =  1.2012311186486965E-001
-0.1909232  0.3583396 -0.4047793  0.3145640 -0.1181507
Generalized matrix eigensolution residuals
 9.44E-016  9.16E-016  1.55E-015  1.44E-015  1.67E-015
Sum of squared residuals = 9.0017965600659404E-030
Rayleigh quotient =  1.2012311186486951E-001
Generalized matrix eigensolution residuals
 9.16E-016  1.03E-015  1.39E-015  1.44E-015  1.67E-015
Sum of squared residuals = 8.6759292134756201E-030
```

Algorithm 25 – Rayleight quotient minimization

Fortran

Listing

```
C&&& A25
C  TEST ALG 25 USING GRID (5 POINT)
C  J.C. NASH   JULY 1978, APRIL 1989
      LOGICAL IFR
      INTEGER N,M,NOUT,NIN,KPR,LIMIT,I
      EXTERNAL APR,BPR
C     REAL EPS,P0,X(N),S(N),T(N),U(N),V(N),W(N),Y(N),RNORM
      COMMON /GSZ/ M,IFR,R(1600)
      REAL EPS,P0,RNORM,VNORM,RNV
      REAL S(1600),T(1600),U(1600),V(1600),W(1600),X(1600),Y(1600)
C  I/O CHANNELS
      NIN=5
      NOUT=6
  1   READ(NIN,900)M,LIMIT
 900  FORMAT(2I4)
      N=M*M
      WRITE(NOUT,950)M,N,LIMIT
 950  FORMAT(' GRID ORDER',I4,' EQNS ORDER',I5,'  LIMIT=',I4)
      IF(M.LE.0)STOP
      IFR=.FALSE.
C  IBM MACHINE PRECISION
      EPS=16.0**(-5)
      KPR=LIMIT
      RNORM=1.0/SQRT(FLOAT(N))
      DO 10 I=1,N
        X(I)=RNORM
  10  CONTINUE
      CALL A25RQM(N,X,EPS,KPR,S,T,U,V,W,Y,P0,NOUT,APR,BPR)
      WRITE(NOUT,951)KPR,P0
 951  FORMAT(' RETURNED AFTER',I4,' PRODUCTS WITH EV=',1PE16.8)
      DO 20 I=1,N
        R(I)=-P0*X(I)
  20  CONTINUE
      CALL APR(N,X,V)
      RNORM=0.0
      VNORM=0.0
      DO 30 I=1,N
        RNORM=RNORM+(V(I)+R(I))**2
        VNORM=VNORM+X(I)**2
  30  CONTINUE
      RNORM=SQRT(RNORM/N)
      VNORM=SQRT(VNORM/N)
      RNV=RNORM/VNORM
      WRITE(NOUT,952)RNORM,VNORM,RNV
 952  FORMAT(' RESIDUAL NORM=',1PE16.8,' /',E16.8,'=',E16.8)
      GOTO 1
      END
      SUBROUTINE BPR(N,X,V)
```

```
C  J.C. NASH   JULY 1978, APRIL 1989
C  UNITM MATRIX * X  INTO V
      INTEGER N,I
      REAL X(N),V(N)
      DO 100 I=1,N
        V(I)=X(I)
 100  CONTINUE
      RETURN
      END
      SUBROUTINE APR(N,X,V)
C  J.C. NASH   JULY 1978, APRIL 1989
      LOGICAL IFR
      INTEGER N,I,J,M
      DOUBLE PRECISION S
      REAL X(N),V(N),D,Q
C  M BY M GRID OF A GEORGE  M=SQRT(N)
      COMMON /GSZ/ M,IFR,R(1600)
C     COMMON /GSZ/M
      D=4.0
      Q=-1.0
      DO 100 I=1,N
C  NOTE ALL INTEGERS
      J=I/M
      J=M*J
      S=D*X(I)
C  SUBTRACT RHS FOR RESIDUAL
      IF(IFR)S=S-R(I)
C  LEFT EDGE
      IF(I-J.EQ.1)GOTO 20
      S=S+Q*X(I-1)
C  RIGHT EDGE
  20  IF(I.GT.J)S=S+Q*X(I+1)
C  TOP EDGE
      IF(I.GT.M)S=S+Q*X(I-M)
C  BOTTOM EDGE
      IF(I.LE.N-M)S=S+Q*X(I+M)
      V(I)=S
  100 CONTINUE
      RETURN
      END
      SUBROUTINE A25RQM(N,X,EPS,KPR,Y,Z,T,G,A,B,PO,IPR,APR,BPR)
C  ALGORITHM 25 RAYLEIGH QUOTIENT MINIMIZATION BY CONJUGATE GRADIENTS
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C    N   =  ORDER OF PROBLEM
C    X   =  INITIAL (APPROXIMATE?) EIGENVECTOR
C   EPS  =  MACHINE PRECISION
C&&& for Microsoft test replace with actual names
C   APR,BPR  ARE NAMES OF SUBROUTINES WHICH FORM THE PRODUCTS
C          V= A*X    VIA   CALL APR(N,X,V)
C          T= B*X    VIA   CALL BPR(N,X,T)
C   KPR  =  LIMIT ON THE NUMBER OF PRODUCTS (INPUT) (TAKES ROLE OF IPR)
C        =  PRODUCTS USED (OUTPUT)
C  Y,Z,T,G,A,B RE WORKING VECTORS IN AT LEAST N ELEMENTS
```

```
C   P0   =  APPROXIMATE EIGENVALUE (OUTPUT)
C   IPR  =  PRINT CHANNEL   PRINTING IF IPR.GT.0
C  STEP 0
      INTEGER N,LP,IPR,ITN,I,LIM,COUNT
      REAL X(N),T(N),G(N),Y(N),Z(N),PN,A(N),B(N)
      REAL EPS,TOL,P0,PA,XAX,XBX,XAT,XBT,TAT,TBT,W,K,D,V,GG,BETA,TABT,U
C  IBM VALUE   -  APPROX. LARGEST NUMBER REPRESENTABLE.
C&&&       PA=R1MACH(2)
      PA=1E+35
      LIM=KPR
      KPR=0
      TOL=N*N*EPS*EPS
C  STEP 1
  10  KPR=KPR+1
      IF(KPR.GT.LIM)RETURN
C  FIND LIMIT IN ORIGINAL PROGRAMS
      CALL APR(N,X,A)
      CALL BPR(N,X,B)
C  STEP 2
      XAX=0.0
      XBX=0.0
      DO 25 I=1,N
        XAX=XAX+X(I)*A(I)
        XBX=XBX+X(I)*B(I)
  25  CONTINUE
C  STEP 3
      IF(XBX.LT.TOL)STOP
C  STEP 4
      P0=XAX/XBX
      IF(P0.GE.PA)RETURN
      IF(IPR.GT.0)WRITE(IPR,963)KPR,P0
 963  FORMAT( 1H ,I4,' PRODUCTS, EST. EIGENVALUE=',1PE16.8)
C  STEP 5
      PA=P0
C  STEP 6
      GG=0.0
      DO 65 I=1,N
        G(I)=2.0*(A(I)-P0*B(I))/XBX
        GG=GG+G(I)**2
  65  CONTINUE
C  STEP 7
      IF(IPR.GT.0)WRITE(IPR,964)GG
 964  FORMAT(' GRADIENT NORM SQUARED=',1PE16.8)
      IF(GG.LT.TOL)RETURN
C  STEP 8
      DO 85 I=1,N
        T(I)=-G(I)
  85  CONTINUE
C  STEP 9
      DO 240 ITN=1,N
C  STEP 10
        KPR=KPR+1
        IF(KPR.GT.LIM)RETURN
```

```
      CALL APR(N,T,Y)
      CALL BPR(N,T,Z)
C  STEP 11
      TAT=0.0
      TBT=0.0
      XAT=0.0
      XBT=0.0
      DO 115 I=1,N
      TAT=TAT+T(I)*Y(I)
       XAT=XAT+X(I)*Y(I)
        TBT=TBT+T(I)*Z(I)
       XBT=XBT+X(I)*Z(I)
 115  CONTINUE
C  STEP 12
      U=TAT*XBT-XAT*TBT
      V=TAT*XBX-XAX*TBT
      W=XAT*XBX-XAX*XBT
      D=V*V-4.0*U*W
C  STEP 13
      IF(D.LT.0)STOP
C  MAY NOT WISH TO STOP
C  STEP 14
      D=SQRT(D)
      IF(V.GT.0.0)GOTO 145
      K=0.5*(D-V)/U
      GOTO 150
 145  K=-2.0*W/(D+V)
 150  COUNT=0
C  STEP 15
      XAX=0.0
      XBX=0.0
      DO 155 I=1,N
        A(I)=A(I)+K*Y(I)
        B(I)=B(I)+K*Z(I)
        W=X(I)
        X(I)=W+K*T(I)
        IF(W.EQ.X(I))COUNT=COUNT+1
        XAX=XAX+X(I)*A(I)
        XBX=XBX+X(I)*B(I)
 155  CONTINUE
C  STEP 16
      IF(XBX.LT.TOL)STOP
      PN=XAX/XBX
C  STEP 17
      IF(COUNT.LT.N)GOTO 180
      IF(ITN.EQ.1)RETURN
      GOTO 10
C  STEP 18
 180  IF(PN.LT.P0)GOTO 190
      IF(ITN.EQ.1)RETURN
      GOTO 10
C  STEP 19
 190  P0=PN
```

```
        GG=0.0
        DO 195 I=1,N
        G(I)=2.0*(A(I)-PN*B(I))/XBX
        GG=GG+G(I)**2
 195    CONTINUE
C  STEP 20
        IF(GG.LT.TOL)GOTO 10
C  STEP 21
        XBT=0.0
        DO 215 I=1,N
         XBT=XBT+X(I)*Z(I)
 215    CONTINUE
C  STEP 22
        TABT=0.0
        BETA=0.0
        DO 225 I=1,N
          W=Y(I)-PN*Z(I)
          TABT=TABT+T(I)*W
          BETA=BETA+G(I)*(W-G(I)*XBT)
 225    CONTINUE
C  STEP 23
        BETA=BETA/TABT
        DO 235 I=1,N
          T(I)=BETA*T(I)-G(I)
 235    CONTINUE
C  STEP 24
 240  CONTINUE
C  STEP 25
      GOTO 10
C  NO STEP 26  -  HAVE USED RETURN INSTEAD
      END
```

Example output

?? explanation needed

```
gfortran ../fortran/a25.f
mv ./a.out ../fortran/a25.run
../fortran/a25.run < ../fortran/a25.in
```

```
##  GRID ORDER   3 EQNS ORDER    9  LIMIT= 100
##     1 PRODUCTS, EST. EIGENVALUE=  1.33333337E+00
##  GRADIENT NORM SQUARED=  1.77777791E+00
##     5 PRODUCTS, EST. EIGENVALUE=  1.17157304E+00
##  GRADIENT NORM SQUARED=  5.56937471E-13
##  RETURNED AFTER   5 PRODUCTS WITH EV=  1.17157304E+00
##  RESIDUAL NORM=  1.31790827E-07 /  3.43119442E-01=  3.84096069E-07
##  GRID ORDER  10 EQNS ORDER  100  LIMIT= 400
##     1 PRODUCTS, EST. EIGENVALUE=  4.00000244E-01
##  GRADIENT NORM SQUARED=  1.28000033E+00
##    15 PRODUCTS, EST. EIGENVALUE=  1.62028164E-01
##  GRADIENT NORM SQUARED=  7.54772778E-09
##  RETURNED AFTER  15 PRODUCTS WITH EV=  1.62028164E-01
##  RESIDUAL NORM=  5.59246428E-06 /  1.13465175E-01=  4.92879371E-05
##  GRID ORDER  -1 EQNS ORDER    1  LIMIT=   0
```

Pascal

Listing

```
program dr25(input, output);
{dr25.pas == eigensolutions by minimisation of the Rayleigh quotient

          Copyright 1988 J.C.Nash
}
{constype.def ==
  This file contains various definitions and type statements which are
  used throughout the collection of "Compact Numerical Methods".  In many
  cases not all definitions are needed, and users with very tight memory
  constraints may wish to remove some of the lines of this file when
  compiling certain programs.

  Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}
uses Dos, Crt; {Turbo Pascal 5.0 Modules}
{ 1. Interrupt, Unit, Interface, Implementation, Uses are reserved words now.}
{ 2. System,Dos,Crt are standard unit names in Turbo 5.0.}

const
  big = 1.0E+35;    {a very large number}
  Maxconst = 25;    {Maximum number of constants in data record}
  Maxobs = 100;     {Maximum number of observations in data record}
  Maxparm  = 25;    {Maximum number of parameters to adjust}
  Maxvars = 10;     {Maximum number of variables in data record}
  acctol = 0.0001;  {acceptable point tolerance for minimisation codes}
  maxm = 20;        {Maximum number or rows in a matrix}
  maxn = 20;        {Maximum number of columns in a matrix}
  maxmn = 40;       {maxn+maxm, the number of rows in a working array}
  maxsym = 210;     {maximum number of elements of a symmetric matrix
              which need to be stored = maxm * (maxm + 1)/2 }
  reltest = 10.0;   {a relative size used to check equality of numbers.
              Numbers x and y are considered equal if the
              floating-point representation of reltest+x equals
              that of reltest+y.}
  stepredn = 0.2;   {factor to reduce stepsize in line search}
  yearwrit = 1990;  {year in which file was written}

type
  str2 = string[2];
  rmatrix = array[1..maxm, 1..maxn] of real; {a real matrix}
  wmatrix = array[1..maxmn, 1..maxn] of real; {a working array, formed
                  as one real matrix stacked on another}
  smatvec = array[1..maxsym] of real; {a vector to store a symmetric matrix
              as the row-wise expansion of its lower triangle}
  rvector = array[1..maxm] of real;  {a real vector. We will use vectors
              of m elements always. While this is NOT space efficient,
              it simplifies program codes.}
  cgmethodtype= (Fletcher_Reeves,Polak_Ribiere,Beale_Sorenson);
```

```
    {three possible forms of the conjugate gradients updating formulae}
  probdata = record
          m      : integer; {number of observations}
          nvar   : integer; {number of variables}
          nconst: integer; {number of constants}
          vconst: array[1..Maxconst] of real;
          Ydata : array[1..Maxobs, 1..Maxvars] of real;
          nlls   : boolean; {true if problem is nonlinear least squares}
        end;
{
  NOTE: Pascal does not let us define the work-space for the function
  within the user-defined code.  This is a weakness of Pascal for this
  type of work.
}
var {global definitions}
  banner     : string[80]; {program name and description}

function calceps:real;
{calceps.pas ==
  This function returns the machine EPSILON or floating point tolerance,
  the smallest positive real number such that 1.0 + EPSILON > 1.0.
  EPSILON is needed to set various tolerances for different algorithms.
  While it could be entered as a constant, I prefer to calculate it, since
  users tend to move software between machines without paying attention to
  the computing environment. Note that more complete routines exist.
}
var
  e,e0: real;
  i: integer;
begin {calculate machine epsilon}
  e0 := 1; i:=0;
  repeat
    e0 := e0/2; e := 1+e0;  i := i+1;
  until (e=1.0) or (i=50); {note safety check}
  e0 := e0*2;
{ Writeln('Machine EPSILON =',e0);}
  calceps:=e0;
end; {calceps}

{$I matrixin.pas}
{$I vectorin.pas}

procedure matmul(nn : integer; {order of matrix}
            matrix: rmatrix;{the matrix or order nn}
            vectorin: rvector;{vector which is multiplied}
          var vectorout: rvector); {product vector}
{matmul.pas == Here we use an explicit matrix multiplication. This may
  be replaced by implicit forms as appropriate.}
var
  ii, jj : integer;
  tt : real;
begin
  for ii := 1 to nn do
```

```
  begin
    tt := 0.0;
    for jj := 1 to nn do tt := tt+matrix[ii,jj]*vectorin[jj];
    vectorout[ii] := tt;
  end; {loop on ii}
end; {matmul.pas}


procedure rqmcg( n : integer;
            A, B : rmatrix;
          var X : rvector;
          var ipr : integer;
          var rq : real);

var
  count, i, itn, itlimit : integer;
  avec, bvec, yvec, zvec, g, t : rvector;
  beta, d, eps, g2, gg, oldg2, pa, pn, s2, step : real;
  t2, ta, tabt, tat, tb, tbt, tol, u, v, w, xat, xax, xbt, xbx : real;
  conv, fail : boolean;

begin
  writeln('alg25.pas -- Rayleigh quotient minimisation');
  itlimit := ipr;
  fail := false;
  conv := false;
  ipr := 0;
  eps := calceps;
  tol := n*n*eps*eps;

  pa := big;
  while (ipr<=itlimit) and (not conv) do
  begin
    matmul(n, A, X, avec);
    matmul(n, B, X, bvec);
    ipr := ipr+1;

    xax := 0.0; xbx := 0.0;
    for i := 1 to n do
    begin
      xax := xax+X[i]*avec[i]; xbx := xbx+X[i]*bvec[i];
    end;
    if xbx<=tol then halt;
    rq := xax/xbx;
    write(ipr,' products -- ev approx. =',rq:18);
    if rq<pa then
    begin
      pa := rq;
      gg := 0.0;
      for i := 1 to n do
      begin
        g[i] := 2.0*(avec[i]-rq*bvec[i])/xbx; gg := gg+g[i]*g[i];
      end;
```

```
    writeln(' squared gradient norm =',gg:8);
    if gg>tol then

    begin

      for i := 1 to n do t[i] := -g[i];
      itn := 0;
      repeat
        itn := itn+1;
        matmul(n, A, t, yvec);
        matmul(n, B, t, zvec); ipr := ipr+1;
        tat := 0.0; tbt := 0.0; xat := 0.0; xbt := 0.0;
        for i := 1 to n do
        begin
          xat := xat+X[i]*yvec[i]; tat := tat+t[i]*yvec[i];
          xbt := xbt+X[i]*zvec[i]; tbt := tbt+t[i]*zvec[i];
        end;

        u := tat*xbt-xat*tbt; v := tat*xbx-xax*tbt;
        w := xat*xbx-xax*xbt; d := v*v-4.0*u*w;
        if d<0.0 then halt;

        d := sqrt(d);
        if v>0.0 then step := -2.0*w/(v+d) else step := 0.5*(d-v)/u;

        count := 0;
        xax := 0.0; xbx := 0.0;
        for i := 1 to n do
        begin
          avec[i] := avec[i]+step*yvec[i];
          bvec[i] := bvec[i]+step*zvec[i];
          w := X[i]; X[i] := w+step*t[i];
          if (reltest+w)=(reltest+X[i]) then count := count+1;
          xax := xax+X[i]*avec[i]; xbx := xbx+X[i]*bvec[i];
        end;
        if xbx<=tol then halt
                else pn := xax/xbx;
        if (count<n) and (pn<rq) then
        begin
          rq := pn; gg := 0.0;
          for i := 1 to n do
          begin
            g[i] := 2.0*(avec[i]-pn*bvec[i])/xbx; gg := gg+g[i]*g[i];
          end;
          if gg>tol then
          begin
            xbt := 0.0; for i := 1 to n do xbt := xbt+X[i]*zvec[i];

            tabt := 0.0; beta := 0.0;
            for i := 1 to n do
            begin
              w := yvec[i]-pn*zvec[i]; tabt := tabt+t[i]*w;
              beta := beta+g[i]*(w-g[i]*xbt);
```

```
              end;
              beta := beta/tabt;

              for i := 1 to n do t[i] := beta*t[i]-g[i];
            end;
          end

          else
          begin
            if itn=1 then conv := true;
            itn := n+1;
          end;
        until (itn>=n) or (count=n) or (gg<=tol) or conv;
      end
      else conv := true;
    end
    else
    begin
      conv := true;
    end;
    ta := 0.0;
    for i := 1 to n do ta := ta+sqr(X[i]); ta := 1.0/sqrt(ta);
    for i := 1 to n do X[i] := ta*X[i];
  end;
  if ipr>itlimit then ipr := -ipr;
  writeln;
end;

function resids(nRow, nCol: integer; A : rmatrix;
          Y: rvector; Bvec : rvector; print : boolean):real;
{resids.pas
  == Computes residuals and , if print is TRUE, displays them 7
    per line for the linear least squares problem. The sum of
    squared residuals is returned.

    residual vector = A * Bvec - Y
}
var
i, j: integer;
t1, ss : real;

begin
  if print then
  begin
    writeln('Residuals');
  end;
  ss:=0.0;
  for i:=1 to nRow do
  begin
    t1:=-Y[i]; {note form of residual is residual = A * B - Y }
    for j:=1 to nCol do
      t1:=t1+A[i,j]*Bvec[j];
    ss:=ss+t1*t1;
```

```
    if print then
    begin
      write(t1:10,' ');
      if (i = 7 * (i div 7)) and (i<nRow) then writeln;
    end;
  end; {loop on i}
  if print then
  begin
    writeln;
    writeln('Sum of squared residuals =',ss);
  end;
  resids:=ss
end; {resids.pas == residual calculation for linear least squares}


var
  A, B : rmatrix;
  X : rvector; {eigenvector}
  Y : rvector; {for residuals}
  avec : smatvec; {for matrixin only}
  sym : boolean; {to tell if matrix symmetric}
  ch : char;
  i, j, n, itcount : integer;
  ev, t, s : real;

begin
  banner:='dr25.pas -- minimise Rayleigh quotient';
  write('Order of problem =');
  readln(n); writeln(n);
  writeln('Matrix A');
??
  matrixin(n, n, A, avec, sym);
  if not sym then
  begin
    writeln('Matrix not symmetric -- halting');
    halt;
  end;
??
  writeln('Metric matrix B');
  matrixin(n, n, B, avec, sym);
??  if not sym then
  begin
    writeln('Matrix not symmetric -- halting');
    writeln(confile,'Matrix not symmetric -- halting');
    halt;
  end;
??
  writeln('Initial eigenvector approximation');
??  vectorin(n, X);
  itcount:=100*n; {safety setting}
  rqmcg( n, A, B, X, itcount, ev);
  writeln('Solution after ',itcount,' products. Est. eigenvalue =',ev);
  for i:=1 to n do
```

```
  begin
    write(X[i]:10:7,' ');
    if (7 * (i div 7) = i) and (i<n) then  writeln;
    t:=0.0;
    for j:=1 to n do t:=t+B[i,j]*X[j];
    Y[i]:=ev*t; {to save eigenvalue * matrix-vector product for residuals}
  end;
  writeln;
  s := resids(n, n, A, Y, X, true);
end. {dr25.pas}
```

Example output

?? not yet working

Algorithms added in the 2nd Edition, 1990.

Algorithm 26 – Complex matrix eigensolutions

Fortran

Listing

```
C     MASTER COMDRIVE
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER UPP
      INTEGER CLKTC1, CLKTC2
      DIMENSION AR(10,10),AI(10,10),ASR(10,10),ASI(10,10),WR(10),WI(10)
      DIMENSION ZR(10,10),ZI(10,10),SCALE(10),INT(10)
      NIN = 5
      NOUT = 6
    1 READ (NIN,901) N
      IF (N.LE.0) STOP
      WRITE(NOUT, 900)N
 900  FORMAT(' COMPLEX MATRIX OF ORDER ',I4,' REAL FRANK, IMAG MOLER')
      DO 110 I=1,N
         DO 105 J=1,N
            AR(I,J) = MIN(I,J)
            AI(I,J) = MIN(I,J) - 2.0
  105    CONTINUE
         AR(I,I)=I
         AI(I,I)=I
  110  CONTINUE
C   START
      RGRD=0.0
      ISS=0
      JSS=0
      WRITE (NOUT,903) N
      WRITE (NOUT,904)
      do 103 i=1,n
      WRITE (NOUT,905) (AR(I,J),J=1,N)
 103  continue
      WRITE (NOUT,906)
```

```
      do 104 i=1,n
      WRITE (NOUT,905) (AI(I,J),J=1,N)
 104  continue
      DO 25 I=1,N
      DO 20 J=1,N
      ASR(I,J)=AR(I,J)
      ASI(I,J)=AI(I,J)
 20   CONTINUE
 25   CONTINUE
      RADX=16.0
c       CALL TIMER(CLKTC1)
c   20210128 -- SUPPRESS BALANCING FOR NOW
C       CALL CBAL(10,N,RADX,AR,AI,LOW,LUP,SCALE)
      CALL COMEIG(N,10,AR,AI,ZR,ZI,WR,WI)
C       CALL CBABK2(10,N,LOW,LUP,SCALE,N,ZR,ZI)
c       CALL TIMER(CLKTC2)
C       CLKTC2=CLKTC2-CLKTC1
      DO 6 J=1,N
      IT=1
      BIG=ZR(1,J)**2+ZI(1,J)**2
      IF (N.LT.2) GO TO 4
      DO 3 I=2,N
      U=ZR(I,J)**2+ZI(I,J)**2
      IF (U.LE.BIG) GO TO 3
      BIG=U
      IT=I
    3 CONTINUE
    4 U=ZR(IT,J)/BIG
      V=-ZI(IT,J)/BIG
      DO 5 I=1,N
      BIG=ZR(I,J)*U-ZI(I,J)*V
      ZI(I,J)=ZI(I,J)*U+ZR(I,J)*V
      ZR(I,J)=BIG
    5 CONTINUE
    6 CONTINUE
      DO 9 J=1,N
      WRITE (NOUT,907) J,WR(J),WI(J)
      WRITE (NOUT,908) (ZR(I,J),ZI(I,J),I=1,N)
      WRITE (NOUT,909)
      AL=WR(J)
      GA=WI(J)
      DO 8 I=1,N
      U=0.0
      V=0.0
      DO 7 K=1,N
      U=U+ASR(I,K)*ZR(K,J)-ASI(I,K)*ZI(K,J)
      V=V+ASR(I,K)*ZI(K,J)+ASI(I,K)*ZR(K,J)
    7 CONTINUE
      U=U-AL*ZR(I,J)+GA*ZI(I,J)
      V=V-GA*ZR(I,J)-AL*ZI(I,J)
      TEM=DSQRT(U**2+V**2)
      IF (TEM.LE.RGRD) GO TO 8
      RGRD=TEM
```

```
      ISS=I
      JSS=J
      WRITE (NOUT,908) U,V
    8 CONTINUE
    9 CONTINUE
      WRITE (NOUT,910) RGRD,JSS,ISS
C      WRITE (NOUT,912) CLKTC2
      GO TO 1
  901 FORMAT (I4)
  902 FORMAT (8F10.5)
  903 FORMAT (' ORDER OF MATRIX',I4)
  904 FORMAT (' REAL PART OF MATRIX')
  905 FORMAT (' ',1P5D16.8)
  906 FORMAT (' IMAGINARY PART OF MATRIX')
  907 FORMAT (' EIGENVALUE ',I4,' = ',1PD16.8,' + I*',1PD16.8)
  908 FORMAT (' ',2D16.8)
  909 FORMAT (' RESIDUALS,REAL AND IMAGINARY')
  910 FORMAT (' MAXIMUM RESIDUAL MAGNITUDE=',1PD16.8,' SOLUTION',I4,
     #' ELEMENT',I4)
C  912 FORMAT ('  TIME FOR EIGENSOLUTION =',I9,' * 0.01 SECS')
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     include 'comeig.for'
      SUBROUTINE COMEIG(N,ND,A,Z,T,U,RR,RI)
      IMPLICIT REAL*8(A-H,O-Z)
      LOGICAL MARK
C     SOLVES COMPLEX EIGENPROBLEM FOR A+I*Z
C     VECTORS (RIGHT HAND) RETURNED IN T+I*U
C     EIGENVALUES IN DECREASING ORDER OF MAGNITUDE DOWN DIAGONALS OF A
C     AND Z ARE RESTORED IN ER AND EI ON OUTPUT
C     EPS IS MACHINE DEPENDENT TOLERANCE
C     ITERATION LIMIT IS 35
      DIMENSION A(ND,N),Z(ND,N),T(ND,N),U(ND,N),RR(N),RI(N)
      EQUIVALENCE (AKI,AIK,TIK),(AIM,AMI,TIM)
      EQUIVALENCE (ZKI,ZIK,UIK),(ZMI,ZIM,UIM)
      IF (N.LE.1) GO TO 24
C     SET TOLERANCES
      CALL ENVROD(IB,IT,IR)
      EPS=(1.0D0*IB)**(1-IT)
      MARK=.FALSE.
C     PUT IDENTITY MATRIX IN T AND ZERO IN U
      N1=N-1
      DO 2 I=1,N1
      T(I,I)=1.0D0
      U(I,I)=0.0D0
      I1=I+1
      DO 1 J=I1,N
      T(I,J)=0.0D0
      U(I,J)=0.0D0
      U(J,I)=0.0D0
      T(J,I)=0.0D0
    1 CONTINUE
    2 CONTINUE
```

```
      T(N,N)=1.0D0
      U(N,N)=0.0D0
C     SAFETY LOOP
      DO 23 IT=1,35
      IF (MARK) GO TO 24
C     CONVERGENCE CRITERIA
      TAU=0.0D0
      DO 4 K=1,N
      TEM=0.0D0
      DO 3 I=1,N
      IF (I.NE.K) TEM=DABS(A(I,K))+DABS(Z(I,K))+TEM
    3 CONTINUE
      TAU=TAU+TEM
      RR(K)=TEM+DABS(A(K,K))+DABS(Z(K,K))
    4 CONTINUE
      WRITE (NOUT,901) TAU,IT
C     INTERCHANGE COLUMNS AND ROWS
      DO 8 K=1,N1
      SMAX=RR(K)
      I=K
      K1=K+1
      DO 5 J=K1,N
      IF (SMAX.GE.RR(J)) GO TO 5
      SMAX=RR(J)
      I=J
    5 CONTINUE
      IF (I.EQ.K) GO TO 8
      RR(I)=RR(K)
      DO 6 J=1,N
      TEP=A(K,J)
      A(K,J)=A(I,J)
      A(I,J)=TEP
      TEP=Z(K,J)
      Z(K,J)=Z(I,J)
      Z(I,J)=TEP
    6 CONTINUE
      DO 7 J=1,N
      TEP=A(J,K)
      A(J,K)=A(J,I)
      A(J,I)=TEP
      TEP=Z(J,K)
      Z(J,K)=Z(J,I)
      Z(J,I)=TEP
      TEP=T(J,K)
      T(J,K)=T(J,I)
      T(J,I)=TEP
      TEP=U(J,K)
      U(J,K)=U(J,I)
      U(J,I)=TEP
    7 CONTINUE
    8 CONTINUE
      IF (TAU.LT.(100.0D0*EPS)) GO TO 24
C     BEGIN SWEEP
```

```
      MARK=.TRUE.
      DO 22 K=1,N1
      K1=K+1
      DO 21 M=K1,N
      HJ=0.0D0
      HR=0.0D0
      HI=0.0D0
      G=0.0D0
      DO 9 I=1,N
      IF (I.EQ.K.OR.I.EQ.M) GO TO 9
      HR=HR+A(K,I)*A(M,I)+Z(K,I)*Z(M,I)-A(I,K)*A(I,M)-Z(I,K)*Z(I,M)
      HI=HI+Z(K,I)*A(M,I)-A(K,I)*Z(M,I)-A(I,K)*Z(I,M)+Z(I,K)*A(I,M)
      TE=A(I,K)**2+Z(I,K)**2+A(M,I)**2+Z(M,I)**2
      TEE=A(I,M)**2+Z(I,M)**2+A(K,I)**2+Z(K,I)**2
      G=G+TE+TEE
      HJ=HJ-TE+TEE
    9 CONTINUE
      BR=A(K,M)+A(M,K)
      BI=Z(K,M)+Z(M,K)
      ER=A(K,M)-A(M,K)
      EI=Z(K,M)-Z(M,K)
      DR=A(K,K)-A(M,M)
      DI=Z(K,K)-Z(M,M)
      TE=BR**2+EI**2+DR**2
      TEE=BI**2+ER**2+DI**2
      IF (TE.LT.TEE) GO TO 10
      SISW=1.0D0
      C=BR
      S=EI
      D=DR
      DE=DI
      ROOT2=DSQRT(TE)
      GO TO 11
   10 SISW=-1.0D0
      C=BI
      S=-ER
      D=DI
      DE=DR
      ROOT2=DSQRT(TEE)
   11 ROOT1=DSQRT(S*S+C*C)
      SIG=DSIGN(1.0D0,D)
      SA=0.0D0
      CA=DSIGN(1.0D0,C)
      IF (ROOT1.GE.EPS) GO TO 14
      SX=0.0D0
      SA=0.0D0
      CX=1.0D0
      CA=1.0D0
      IF (SISW.GT.0.0D0) GO TO 12
      E=EI
      B=-BR
      GO TO 13
   12 E=ER
```

```
      B=BI
   13 SND=D**2+DE**2
      GO TO 16
   14 IF (DABS(S).LE.EPS) GO TO 15
      CA=C/ROOT1
      SA=S/ROOT1
   15 COT2X=D/ROOT1
      COTX=COT2X+(SIG*DSQRT(1.0D0+COT2X**2))
      SX=SIG/DSQRT(1.0D0+COTX**2)
      CX=SX*COTX
C     FIND ROTATED ELEMENTS
      ETA=(ER*BR+BI*EI)/ROOT1
      TSE=(BR*BI-ER*EI)/ROOT1
      TE=SIG*(-ROOT1*DE+TSE*D)/ROOT2
      TEE=(D*DE+ROOT1*TSE)/ROOT2
      SND=ROOT2**2+TEE**2
      TEE=HJ*CX*SX
      COS2A=CA**2-SA**2
      SIN2A=2.0D0*CA*SA
      TEM=HR*COS2A+HI*SIN2A
      TEP=HI*COS2A-HR*SIN2A
      HR=CX*CX*HR-SX*SX*TEM-CA*TEE
      HI=CX*CX*HI+SX*SX*TEP-SA*TEE
      B=SISW*TE*CA+ETA*SA
      E=CA*ETA-SISW*TE*SA
   16 S=HR-SIG*ROOT2*E
      C=HI-SIG*ROOT2*B
      ROOT=DSQRT(C*C+S*S)
      IF (ROOT.GE.EPS) GO TO 17
      CB=1.0D0
      CH=1.0D0
      SB=0.0D0
      SH=0.0D0
      GO TO 18
   17 CB=-C/ROOT
      SB=S/ROOT
      TEE=CB*B-E*SB
      SNC=TEE*TEE
      TANH=ROOT/(G+2.0D0*(SNC+SND))
      CH=1.0D0/DSQRT(1.0D0-TANH**2)
      SH=CH*TANH
C     PREPARE FOR TRANSFORMATION
   18 TEM=SX*SH*(SA*CB-SB*CA)
      C1R=CX*CH-TEM
      C2R=CX*CH+TEM
      C1I=-SX*SH*(CA*CB+SA*SB)
      C2I=C1I
      TEP=SX*CH*CA
      TEM=CX*SH*SB
      S1R=TEP-TEM
      S2R=-TEP-TEM
      TEP=SX*CH*SA
      TEM=CX*SH*CB
```

```
      S1I=TEP+TEM
      S2I=TEP-TEM
C     DECIDE WHETHER TO MAKE TRANSFORMATION
      TEM=DSQRT(S1R**2+S1I**2)
      TEP=DSQRT(S2R**2+S2I**2)
      IF (TEM.LE.EPS.AND.TEP.LE.EPS) GO TO 21
      MARK=.FALSE.
C     TRANSFORMATION ON LEFT
      DO 19 I=1,N
      AKI=A(K,I)
      AMI=A(M,I)
      ZKI=Z(K,I)
      ZMI=Z(M,I)
      A(K,I)=C1R*AKI-C1I*ZKI+S1R*AMI-S1I*ZMI
      Z(K,I)=C1R*ZKI+C1I*AKI+S1R*ZMI+S1I*AMI
      A(M,I)=S2R*AKI-S2I*ZKI+C2R*AMI-C2I*ZMI
      Z(M,I)=S2R*ZKI+S2I*AKI+C2R*ZMI+C2I*AMI
   19 CONTINUE
C     TRANSFORMATION ON RIGHT
      DO 20 I=1,N
      AKI=A(I,K)
      AMI=A(I,M)
      ZKI=Z(I,K)
      ZMI=Z(I,M)
      A(I,K)=C2R*AKI-C2I*ZKI-S2R*AMI+S2I*ZMI
      Z(I,K)=C2R*ZKI+C2I*AKI-S2R*ZMI-S2I*AMI
      A(I,M)=-S1R*AKI+S1I*ZKI+C1R*AMI-C1I*ZMI
      Z(I,M)=-S1R*ZKI-S1I*AKI+C1R*ZMI+C1I*AMI
      AKI=T(I,K)
      AMI=T(I,M)
      ZKI=U(I,K)
      ZMI=U(I,M)
      T(I,K)=C2R*AKI-C2I*ZKI-S2R*AMI+S2I*ZMI
      U(I,K)=C2R*ZKI+C2I*AKI-S2R*ZMI-S2I*AMI
      T(I,M)=-S1R*AKI+S1I*ZKI+C1R*AMI-C1I*ZMI
      U(I,M)=-S1R*ZKI-S1I*AKI+C1R*ZMI+C1I*AMI
   20 CONTINUE
   21 CONTINUE
   22 CONTINUE
   23 CONTINUE
   24 DO 25 I=1,N
      RR(I)=A(I,I)
      RI(I)=Z(I,I)
   25 CONTINUE
      RETURN
  901 FORMAT (' TAU=',D20.10,' AT ITERATION ',I4)
      END
      SUBROUTINE ENVROD(BETA,T,RND)
C     DOUBLE PRECISION MACHINE ENVIRONMENT
C     NO INPUT
C     PARAMETERS:
C        BETA  - MACHINE RADIX
C        T     - NUMBER OF RADIX DIGITS IN WORD (DOUBLE)
```

```
C        RND   - SET TO 1 IF MACHINE ROUNDS
C                SET TO 0 IF MACHINE CHOPS
C     ALL PARAMETERS ARE INTEGERS
C     MACHINE DOUBLE PRECISION I.E. SMALLEST POSITIVE NUMBER SUCH
C     THAT 1. + NUMBER .GT. 1. , IS DBLE(FLOAT(BETA))**(1-T)
      INTEGER BETA,T,RND
      DOUBLE PRECISION A,B,DBLE
      RND=1
      A=2.0D0
      B=2.0D0
    1 IF ((A+1.0D0)-A.NE.1.0D0) GO TO 2
      A=2.0D0*A
      GO TO 1
    2 IF (A+B.NE.A) GO TO 3
      B=2.0D0*B
      GO TO 2
    3 BETA=(A+B)-A
      IF (A+DBLE(FLOAT(BETA-1)).EQ.A) RND=0
      IF (A+DBLE(FLOAT(BETA-1)).EQ.A) RND=0
      T=0
      A=1.0D0
    4 T=T+1
      A=A*DBLE(FLOAT(BETA))
      IF ((A+1.0D0)-A.EQ.1.0D0) GO TO 4
      RETURN
      END
C      include 'cbal.for'
C
C     ------------------------------------------------------------------
C
      SUBROUTINE CBAL (NM,N,RADIX,AR,AI,LOW,UPP,SCALE)
      IMPLICIT REAL*8 (A-H,O-Z)
C
C
      REAL*8 AR(NM,N),AI(NM,N),SCALE(N)
      INTEGER UPP
      LOGICAL NOCONV
      COMPLEX*16 DCMPLX
C
C     THIS SUBROUTINE IS A TRANSLATION OF THE ALGOL PROCEDURE
C     CBALANCE, WHICH IS A COMPLEX VERSION OF  BALANC,
C     NUM. MATH. 13,293-304(1969) BY PARLETT AND REINSCH.
C
C     THIS SUBROUTINE BALANCES A COMPLEX MATRIX AND ISOLATES
C     EIGENVALUES WHENEVER POSSIBLE.
C
C     ON INPUT--
C
C        NM MUST BE SET TO THE ROW DIMENSION OF TWO-DIMENSIONAL
C          ARRAY PARAMETERS AS DECLARED IN THE CALLING PROGRAM
C          DIMENSION STATEMENT,
C
C        N IS THE ORDER OF THE MATRIX,
```

```
C
C        RADIX IS THE BASE OF THE MACHINE FLOATING POINT
C          REPRESENTATION.  FOR SYSTEM/360 THIS IS 16.0,
C
C        AR AND AI ARE ARRAYS CONTAINING THE REAL AND IMAGINARY
C          PARTS, RESPECTIVELY, OF THE ELEMENTS OF THE MATRIX
C          TO BE BALANCED.
C
C     ON OUTPUT--
C
C        AR AND AI CONTAIN THE REAL AND IMAGINARY PARTS,
C          RESPECTIVELY, OF THE ELEMENTS OF THE BALANCED MATRIX,
C
C        LOW, UPP ARE TWO INTEGERS SUCH THAT AR(I,J) AND AI(I,J)
C          ARE EQUAL TO ZERO IF
C           (1) I IS GREATER THAN J AND
C           (2) J=1,...,LOW-1 OR I=UPP+1,...,N,
C
C        SCALE IS AN ARRAY WHICH CONTAINS INFORMATION DETERMINING
C           THE PERMUTATIONS USED AND SCALING FACTORS.
C
C     SUPPOSE THAT THE PRINCIPAL SUBMATRIX IN ROWS LOW THROUGH UPP
C     HAS BEEN BALANCED, THAT P(J) DENOTES THE INDEX INTERCHANGED
C     WITH J DURING THE PERMUTATION STEP, AND THAT THE ELEMENTS
C     OF THE DIAGONAL MATRIX USED ARE DENOTED BY D(I,J).  THEN
C        SCALE(J) = P(J),    FOR J = 1,...,LOW-1
C                 = D(J,J)       J = LOW,...,UPP
C                 = P(J)         J = UPP+1,...,N.
C     THE ORDER IN WHICH THE INTERCHANGES ARE MADE IS N TO UPP+1,
C     THEN 1 TO LOW-1.
C
C     THE ALGOL PROCEDURE EXC CONTAINED IN CBALANCE APPEARS IN
C     CBAL  IN LINE.  (NOTE THAT THE ALGOL ROLES OF IDENTIFIERS
C     K,L HAVE BEEN REVERSED.)
C
C     ARITHMETIC IS REAL EXCEPT FOR THE USE OF THE SUBROUTINES
C     CABS AND CMPLX.
C
C     TRANSLATED BY V. KLEMA, ARGONNE NATIONAL LABORATORY, AUG., 1969.
C     MODIFIED BY B. GARBOW, JAN., 1971.
C
C     ----------------------------------------------------------------
C
      B2 = RADIX * RADIX
      K = 1
      L = N
      GO TO 100
C     ********** IN-LINE PROCEDURE FOR ROW AND
C                COLUMN EXCHANGE **********
   20 SCALE(M) = J
      IF (J .EQ. M) GO TO 50
C
      DO 30 I = 1, L
```

```
         F = AR(I,J)
         AR(I,J) = AR(I,M)
         AR(I,M) = F
         F = AI(I,J)
         AI(I,J) = AI(I,M)
         AI(I,M) = F
   30 CONTINUE
C
      DO 40 I = K, N
         F = AR(J,I)
         AR(J,I) = AR(M,I)
         AR(M,I) = F
         F = AI(J,I)
         AI(J,I) = AI(M,I)
         AI(M,I) = F
   40 CONTINUE
C
   50 GO TO (80,130), IEXC
C     ********** SEARCH FOR ROWS ISOLATING AN EIGENVALUE
C                AND PUSH THEM DOWN **********
   80 L = L - 1
      IF (L .LT. 1) GO TO 280
  100 LP1 = L + 1
C     ********** FOR J=L STEP -1 UNTIL 1 DO -- **********
      DO 120 JJ = 1, L
         J = LP1 - JJ
         R = 0.0
C
         DO 110 I = 1, L
            IF (I .EQ. J) GO TO 110
            R = R + CDABS(DCMPLX(AR(J,I),AI(J,I)))
  110    CONTINUE
C
         IF (R .NE. 0.0) GO TO 120
         M = L
         IEXC = 1
         GO TO 20
  120 CONTINUE
C
      GO TO 140
C     ********** SEARCH FOR COLUMNS ISOLATING AN EIGENVALUE
C                AND PUSH THEM LEFT **********
  130 K = K + 1
C
  140 DO 170 J = K, L
         C = 0.0
C
         DO 150 I = K, L
            IF (I .EQ. J) GO TO 150
            C = C + CDABS(DCMPLX(AR(I,J),AI(I,J)))
  150    CONTINUE
C
         IF (C .NE. 0.0) GO TO 170
```

```
         M = K
         IEXC = 2
         GO TO 20
  170 CONTINUE
C     ********** NOW BALANCE THE SUBMATRIX IN ROWS K TO L **********
      DO 180 I = K, L
      SCALE(I) = 1.0
  180 CONTINUE
C     ********** ITERATIVE LOOP FOR NORM REDUCTION **********
  190 NOCONV = .FALSE.
C
      DO 270 I = K, L
         C = 0.0
         R = 0.0
C
         DO 200 J = K, L
            IF (J .EQ. I) GO TO 200
            C = C + CDABS(DCMPLX(AR(J,I),AI(J,I)))
            R = R + CDABS(DCMPLX(AR(I,J),AI(I,J)))
  200    CONTINUE
C
         G = R / RADIX
         F = 1.0
  210    IF (C .GE. G) GO TO 220
         F = F * RADIX
         C = C * B2
         GO TO 210
  220    G = R * RADIX
  230    IF (C .LT. G) GO TO 240
         F = F / RADIX
         C = C / B2
         GO TO 230
C     ********** NOW BALANCE **********
  240    IF (F .EQ. 1.0) GO TO 270
         G = 1.0 / F
         SCALE(I) = SCALE(I) * F
         NOCONV = .TRUE.
C
         DO 250 J = K, N
            AR(I,J) = AR(I,J) * G
            AI(I,J) = AI(I,J) * G
  250    CONTINUE
C
         DO 260 J = 1, L
            AR(J,I) = AR(J,I) * F
            AI(J,I) = AI(J,I) * F
  260    CONTINUE
C
  270 CONTINUE
C
      IF(NOCONV) GO TO 190
  280 LOW = K
      UPP = L
```

```
      RETURN
C     ********** LAST CARD OF CBAL **********
      END
C      include 'cbabk2.for'
C
C     ------------------------------------------------------------------
C
      SUBROUTINE CBABK2(NM,N,LOW,UPP,SCALE,M,ZR,ZI)
      IMPLICIT REAL*8 (A-H,O-Z)
C
C
      REAL*8 SCALE(N),ZR(NM,M),ZI(NM,M)
      INTEGER UPP
C
C     THIS SUBROUTINE IS A TRANSLATION OF THE ALGOL PROCEDURE
C     CBABK2, WHICH IS A COMPLEX VERSION OF BALBAK,
C     NUM. MATH. 13,293-304(1969) BY PARLETT AND REINSCH.
C
C     THIS SUBROUTINE FORMS THE EIGENVECTORS OF A COMPLEX
C     GENERAL MATRIX BY BACK TRANSFORMING THOSE OF THE
C     CORRESPONDING BALANCED MATRIX PRODUCED BY  CBAL.
C
C     ON INPUT--
C
C        NM MUST BE SET TO THE ROW DIMENSION OF TWO-DIMENSIONAL
C          ARRAY PARAMETERS AS DECLARED IN THE CALLING PROGRAM
C          DIMENSION STATEMENT,
C
C        N IS THE ORDER OF THE MATRIX,
C
C        LOW AND UPP ARE INTEGERS PRODUCED BY THE BALANCING
C          SUBROUTINE  CBAL,
C
C        SCALE IS AN ARRAY PRODUCED BY  CBAL  CONTAINING
C          INFORMATION ABOUT THE PERMUTATION AND SCALING
C          TRANSFORMATIONS USED IN BALANCING,
C
C        M IS AN INTEGER GIVING THE NUMBER OF COLUMNS OF
C          Z = (ZR,ZI) TO BE BACK TRANSFORMED,
C
C        ZR AND ZI ARE ARRAYS, THE FIRST M COLUMNS OF WHICH
C          CONTAIN THE REAL AND IMAGINARY PARTS, RESPECTIVELY,
C          OF THE EIGENVECTORS TO BE BACK TRANSFORMED.
C
C     ON OUTPUT--
C
C        ZR AND ZI CONTAIN THE REAL AND IMAGINARY PARTS,
C          RESPECTIVELY, OF THE BACK TRANSFORMED EIGENVECTORS
C          IN THEIR FIRST M COLUMNS.
C
C     TRANSLATED BY V. KLEMA, ARGONNE NATIONAL LABORATORY, AUG., 1969.
C     MODIFIED BY B. GARBOW, JAN., 1971.
C
```

```
C     ------------------------------------------------------------------
C
      IF (UPP .LT. LOW) GO TO 120
C
      DO 110 I = LOW, UPP
         S = SCALE(I)
C     ********** LEFT HAND EIGENVECTORS ARE BACK TRANSFORMED
C                IF THE FOREGOING STATEMENT IS REPLACED BY
C                S=1.0/SCALE(I) **********
         DO 100 J = 1, M
            ZR(I,J) = ZR(I,J) * S
            ZI(I,J) = ZI(I,J) * S
  100    CONTINUE
C
  110 CONTINUE
C     ********** FOR I=LOW-1 STEP -1 UNTIL 1,
C                UPP+1 STEP 1 UNTIL N DO -- **********
  120 DO 140 II = 1, N
         I = II
         IF (I .GE. LOW .AND. I .LE. UPP) GO TO 140
         IF (I .LT. LOW) I = LOW - II
         K = SCALE(I)
         IF (K .EQ. I) GO TO 140
C
         DO 130 J = 1, M
            S = ZR(I,J)
            ZR(I,J) = ZR(K,J)
            ZR(K,J) = S
            S = ZI(I,J)
            ZI(I,J) = ZI(K,J)
            ZI(K,J) = S
  130    CONTINUE
C
  140 CONTINUE
C
      RETURN
C     ********** LAST CARD OF CBABK2 **********
      END
```

Example output

?? explananation needed. Note use of Pascal input file

```
gfortran ../fortran/comeigd26.f
mv ./a.out ../fortran/comeigd26.run
../fortran/comeigd26.run < ../fortran/d26f.in
```

```
##  COMPLEX MATRIX OF ORDER    1 REAL FRANK, IMAG MOLER
##  ORDER OF MATRIX   1
##  REAL PART OF MATRIX
##    1.00000000D+00
##  IMAGINARY PART OF MATRIX
##    1.00000000D+00
##  EIGENVALUE    1 =   1.00000000D+00 + I*  1.00000000D+00
```

```
##         Infinity            NaN
##  RESIDUALS,REAL AND IMAGINARY
##              NaN            NaN
##  MAXIMUM RESIDUAL MAGNITUDE=             NaN SOLUTION   1 ELEMENT   1
##  COMPLEX MATRIX OF ORDER    5 REAL FRANK, IMAG MOLER
##  ORDER OF MATRIX   5
##  REAL PART OF MATRIX
##    1.00000000D+00  1.00000000D+00  1.00000000D+00  1.00000000D+00  1.00000000D+00
##    1.00000000D+00  2.00000000D+00  2.00000000D+00  2.00000000D+00  2.00000000D+00
##    1.00000000D+00  2.00000000D+00  3.00000000D+00  3.00000000D+00  3.00000000D+00
##    1.00000000D+00  2.00000000D+00  3.00000000D+00  4.00000000D+00  4.00000000D+00
##    1.00000000D+00  2.00000000D+00  3.00000000D+00  4.00000000D+00  5.00000000D+00
##  IMAGINARY PART OF MATRIX
##    1.00000000D+00 -1.00000000D+00 -1.00000000D+00 -1.00000000D+00 -1.00000000D+00
##   -1.00000000D+00  2.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
##   -1.00000000D+00  0.00000000D+00  3.00000000D+00  1.00000000D+00  1.00000000D+00
##   -1.00000000D+00  0.00000000D+00  1.00000000D+00  4.00000000D+00  2.00000000D+00
##   -1.00000000D+00  0.00000000D+00  1.00000000D+00  2.00000000D+00  5.00000000D+00
##  TAU=    0.5600000000D+02 AT ITERATION    1
##  TAU=    0.7210750156D+01 AT ITERATION    2
##  TAU=    0.1616763195D+01 AT ITERATION    3
##  TAU=    0.1413590986D+00 AT ITERATION    4
##  TAU=    0.4138739759D-03 AT ITERATION    5
##  TAU=    0.2340651578D-07 AT ITERATION    6
##  TAU=    0.3311244821D-13 AT ITERATION    7
##  TAU=    0.1320750645D-14 AT ITERATION    8
##  EIGENVALUE    1 =   1.14382331D+01 + I*  5.89078700D+00
##    0.72919929D-01 -0.36498261D+00
##    0.41137353D+00 -0.25750477D+00
##    0.69331293D+00 -0.14425542D+00
##    0.89498614D+00 -0.51704965D-01
##    0.10000000D+01  0.00000000D+00
##  RESIDUALS,REAL AND IMAGINARY
##   -0.22204460D-15 -0.39968029D-14
##    0.37747583D-14 -0.31086245D-14
##    0.56066263D-14 -0.42188475D-14
##  EIGENVALUE    2 =   1.95778803D+00 + I*  1.84957614D+00
##    0.10000000D+01  0.00000000D+00
##    0.53123189D+00  0.45320234D+00
##    0.61249199D-01  0.40348153D+00
##   -0.21682241D+00  0.13113052D+00
##   -0.32155273D+00 -0.84132303D-01
##  RESIDUALS,REAL AND IMAGINARY
##  EIGENVALUE    3 =   9.16755008D-01 + I*  2.61387519D+00
##   -0.71977203D+00 -0.44570949D+00
##    0.52009034D+00 -0.42594831D+00
##    0.10000000D+01 -0.69388939D-17
##    0.22249390D+00  0.17713194D+00
##   -0.79070621D+00  0.76175261D-01
##  RESIDUALS,REAL AND IMAGINARY
##   -0.88817842D-15  0.83821838D-14
##  EIGENVALUE    4 =   4.07280150D-01 + I*  2.37082457D+00
##   -0.48556655D+00 -0.27303323D+00
##    0.10000000D+01  0.13877788D-16
```

```
##   -0.79151794D-01  0.15287163D+00
##   -0.93693223D+00 -0.76818410D-01
##    0.59902399D+00  0.30923994D-02
##  RESIDUALS,REAL AND IMAGINARY
##  EIGENVALUE    5 =   2.79943756D-01 + I*  2.27493710D+00
##    0.25776077D+00  0.11786836D+00
##   -0.78526602D+00 -0.57560560D-01
##    0.10000000D+01  0.00000000D+00
##   -0.81881791D+00  0.25041148D-01
##    0.31426697D+00 -0.13540625D-01
##  RESIDUALS,REAL AND IMAGINARY
##  MAXIMUM RESIDUAL MAGNITUDE=  8.42910830D-15 SOLUTION   3 ELEMENT   1
## Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG IEEE_DIVIDE_BY_ZERO I
```

BASIC

Listing

```
10 DIM A(20,20),Z(20,20),T(20,20),U(20,20),R(20)
15 DIM X(20,20),Y(20,20)
20 LET E9=1e-7 : REM machine precision approximation
25 LET M9=0
30 PRINT "COMEIG AT JULY 31 1984"
40 READ N
45 PRINT "ORDER = ";N
50 IF N <= 0 THEN QUIT
60 LET N1=N-1
80 PRINT "Real part Frank, Imag part Moler"
90 FOR I=1 TO N
100 FOR J=1 TO N
110 IF (I < J) THEN 130
115 LET A(I,J)=J
120 LET Z(I,J)=J-2
125 GOTO 145
130 LET A(I,J)=I
140 LET Z(I,J)=I-2
145 LET Y(I,J)=Z(I,J)
150 LET X(I,J)=A(I,J)
155 LET T(I,J)=0
160 LET U(I,J)=0
170 IF I<>J THEN 190
180 LET T(I,I)=1
190 REM PRINT
200 NEXT J
210 NEXT I
230 LET I3=0
240 LET I3=I3+1
250 IF I3>35 THEN 2220
260 IF M9=1 THEN 2220
270 LET T9=0
280 FOR K=1 TO N
290 LET T8=0
300 FOR I=1 TO N
310 IF I=K THEN 330
```

```
320 LET T8=T8+ABS(A(I,K))+ABS(Z(I,K))
330 NEXT I
340 LET T9=T9+T8
350 LET R(K)=T8+ABS(A(K,K))+ABS(Z(K,K))
360 NEXT K
370 PRINT "TAU=";T9;"  AT ITN ";I3
380 FOR K=1 TO N1
390 LET S9=R(K)
400 LET I=K
410 LET K1=K+1
420 FOR J=K1 TO N
430 IF S9>=R(J) THEN 460
440 LET S9=R(J)
450 LET I=J
460 NEXT J
470 IF I=K THEN 710
480 LET R(I)=R(K)
490 FOR J=1 TO N
500 LET T7=A(K,J)
510 LET A(K,J)=A(I,J)
520 LET A(I,J)=T7
530 LET T7=Z(K,J)
540 LET Z(K,J)=Z(I,J)
550 LET Z(I,J)=T7
560 NEXT J
570 FOR J=1 TO N
580 LET T7=A(J,K)
590 LET A(J,K)=A(J,I)
600 LET A(J,I)=T7
610 LET T7=Z(J,K)
620 LET Z(J,K)=Z(J,I)
630 LET Z(J,I)=T7
640 LET T7=T(J,K)
650 LET T(J,K)=T(J,I)
660 LET T(J,I)=T7
670 LET T7=U(J,K)
680 LET U(J,K)=U(J,I)
690 LET U(J,I)=T7
700 NEXT J
710 NEXT K
720 IF T9<100*E9 THEN 2220
730 LET M9=1
740 FOR K=1 TO N1
750 LET K1=K+1
760 FOR M=K1 TO N
770 LET H1=0
780 LET H2=0
790 LET H3=0
800 LET G=0
810 FOR I=1 TO N
820 IF I=K THEN 920
830 IF I=M THEN 920
840 LET H3=H3+A(K,I)*A(M,I)+Z(K,I)*Z(M,I)
```

```
850 LET H3=H3-A(I,K)*A(I,M)-Z(I,K)*Z(I,M)
860 LET H2=H2+Z(K,I)*A(M,I)-A(K,I)*Z(M,I)
870 LET H2=H2-A(I,K)*Z(I,M)+Z(I,K)*A(I,M)
880 LET T6=A(I,K)*A(I,K)+Z(I,K)*Z(I,K)+A(M,I)*A(M,I)+Z(M,I)*Z(M,I)
890 LET T5=A(I,M)*A(I,M)+Z(I,M)*Z(I,M)+A(K,I)*A(K,I)+Z(K,I)*Z(K,I)
900 LET G=G+T6+T5
910 LET H1=H1-T6+T5
920 NEXT I
930 LET B1=A(K,M)+A(M,K)
940 LET B2=Z(K,M)+Z(M,K)
950 LET E1=A(K,M)-A(M,K)
960 LET E2=Z(K,M)-Z(M,K)
970 LET D1=A(K,K)-A(M,M)
980 LET D2=Z(K,K)-Z(M,M)
990 LET T6=B1*B1+E2*E2+D1*D1
1000 LET T5=B2*B2+E1*E1+D2*D2
1010 IF T6<T5 THEN 1090
1020 LET S8=1
1030 LET C1=B1
1040 LET S=E2
1050 LET C3=D1
1060 LET C4=D2
1070 LET R8=SQR(T6)
1080 GOTO 1150
1090 LET S8=-1
1100 LET C1=B2
1110 LET S=-E1
1120 LET C3=D2
1130 LET C4=D1
1140 LET R8=SQR(T5)
1150 LET R7=SQR(S*S+C1*C1)
1160 LET S6=-1
1170 IF C3<0 THEN 1190
1180 LET S6=1
1190 LET S7=0
1200 LET C7=-1
1210 IF C1<0 THEN 1230
1220 LET C7=1
1230 IF R7>E9 THEN 1360
1240 LET S5=0
1250 LET S7=0
1260 LET C5=1
1270 LET C7=1
1280 IF S8>0 THEN 1320
1290 LET E=E2
1300 LET B=-B1
1310 GOTO 1340
1320 LET E=E1
1330 LET B=B2
1340 LET S4=C3*C3+C4*C4
1350 GOTO 1570
1360 IF ABS(S)<=E9 THEN 1390
1370 LET C7=C1/R7
```

```
1380 LET S7=S/R7
1390 LET C9=C3/R7
1400 LET C0=C9+(S6*SQR(1+C9*C9))
1410 LET S5=S6/SQR(1+C0*C0)
1420 LET C5=S5*C0
1430 LET E3=(E1*B1+E2*B2)/R7
1440 LET T4=(B1*B2-E1*E2)/R7
1450 LET T6=S6*(T4*C3-C4*R7)/R8
1460 LET T5=(C3*C4+R7*T4)/R8
1470 LET S4=R8*R8+T5*T5
1480 LET T5=H1*C5*S5
1490 LET C2=C7*C7-S7*S7
1500 LET S2=2*C7*S7
1510 LET T8=H3*C2+H2*S2
1520 LET T7=H2*C2-H3*S2
1530 LET H3=H3*C5*C5-T8*S5*S5-C7*T5
1540 LET H2=H2*C5*C5+T7*S5*S5-S7*T5
1550 LET B=S8*T6*C7+E3*S7
1560 LET E=C7*E3-S8*T6*S7
1570 LET S=H3-S6*R8*E
1580 LET C1=H2-S6*R8*B
1590 LET R9=SQR(C1*C1+S*S)
1600 IF R9>=E9 THEN 1660
1610 LET C8=1
1620 LET C6=1
1630 LET S0=0
1640 LET S1=0
1650 GOTO 1730
1660 LET C8=-C1/R9
1670 LET S0=S/R9
1680 LET T5=C8*B-E*S0
1690 LET Q8=T5*T5
1700 LET Q9=R9/(G+2*(Q8+S4))
1710 LET C6=1/SQR(1-Q9*Q9)
1720 LET S1=C6*Q9
1730 LET T8=S5*S1*(S7*C8-S0*C7)
1740 LET Q7=C5*C6-T8
1750 LET Q6=C5*C6+T8
1760 LET Q5=-S5*S1*(C7*C8+S7*S0)
1770 LET Q4=Q5
1780 LET T7=S5*C6*C7
1790 LET T8=C5*S1*S0
1800 LET P1=T7-T8
1810 LET P2=-T7-T8
1820 LET T7=S5*C6*S7
1830 LET T8=C5*S1*C8
1840 LET P3=T7+T8
1850 LET P4=T7-T8
1860 LET T8=SQR(P1*P1+P3*P3)
1870 LET T7=SQR(P2*P2+P4*P4)
1880 IF T7>E9 THEN 1900
1890 IF T8<=E9 THEN 2190
1900 LET M9=0
```

```
1910 FOR I=1 TO N
1920 LET P5=A(K,I)
1930 LET P6=A(M,I)
1940 LET P7=Z(K,I)
1950 LET P8=Z(M,I)
1960 LET A(K,I)=Q7*P5-Q5*P7+P1*P6-P3*P8
1970 LET Z(K,I)=Q7*P7+Q5*P5+P1*P8+P3*P6
1980 LET A(M,I)=P2*P5-P4*P7+Q6*P6-Q4*P8
1990 LET Z(M,I)=P2*P7+P4*P5+Q6*P8+Q4*P6
2000 NEXT I
2010 FOR I=1 TO N
2020 LET P5=A(I,K)
2030 LET P6=A(I,M)
2040 LET P7=Z(I,K)
2050 LET P8=Z(I,M)
2060 LET A(I,K)=Q6*P5-Q4*P7-P2*P6+P4*P8
2070 LET Z(I,K)=Q6*P7+Q4*P5-P2*P8-P4*P6
2080 LET A(I,M)=-P1*P5+P3*P7+Q7*P6-Q5*P8
2090 LET Z(I,M)=-P1*P7-P3*P5+Q7*P8+Q5*P6
2100 LET P5=T(I,K)
2110 LET P6=T(I,M)
2120 LET P7=U(I,K)
2130 LET P8=U(I,M)
2140 LET T(I,K)=Q6*P5-Q4*P7-P2*P6+P4*P8
2150 LET U(I,K)=Q6*P7+Q4*P5-P2*P8-P4*P6
2160 LET T(I,M)=-P1*P5+P3*P7+Q7*P6-Q5*P8
2170 LET U(I,M)=-P1*P7-P3*P5+Q7*P8+Q5*P6
2180 NEXT I
2190 NEXT M
2200 NEXT K
2210 GOTO 240
2220 IF I3<=35 THEN 2240
2230 PRINT "FAILURE TO CONVERGE IN 35 ITERATIONS"
2240 PRINT "EIGENSOLUTIONS"
2250 FOR I=1 TO N
2260 LET G=T(1,I)*T(1,I)+U(1,I)*U(1,I)
2270 LET K=1
2280 IF N=1 THEN 2350
2290 FOR M=2 TO N
2300 LET B=T(M,I)*T(M,I)+U(M,I)*U(M,I)
2310 IF B<=G THEN 2340
2320 LET K=M
2330 LET G=B
2340 NEXT M
2350 LET E=T(K,I)/G
2360 LET S=-U(K,I)/G
2370 FOR K=1 TO N
2380 LET G=T(K,I)*E-U(K,I)*S
2390 LET U(K,I)=U(K,I)*E+T(K,I)*S
2400 LET T(K,I)=G
2410 NEXT K
2430 PRINT "EIGENVALUE";I;"=(";A(I,I);",";Z(I,I);")"
2440 PRINT "VECTOR"
```

```
2450 FOR K=1 TO N
2460 PRINT "(";T(K,I);",";U(K,I);")"
2470 NEXT K
2480 PRINT "RESIDUALS"
2490 FOR J=1 TO N
2500 LET S=-A(I,I)*T(J,I)+Z(I,I)*U(J,I)
2510 LET G=-Z(I,I)*T(J,I)-A(I,I)*U(J,I)
2520 FOR K=1 TO N
2530 LET S=S+X(J,K)*T(K,I)-Y(J,K)*U(K,I)
2540 LET G=G+X(J,K)*U(K,I)+Y(J,K)*T(K,I)
2550 NEXT K
2560 PRINT "(";S;",";G;")"
2570 NEXT J
2590 REM
2600 NEXT I
2610 GOTO 40
2620 DATA 1, 5, 0
2800 END
```

Example output

```
bwbasic ../BASIC/comeiga26.bas  >../BASIC/a26.out
# echo "done"
```

```
Bywater BASIC Interpreter/Shell, version 2.20 patch level 2

Copyright (c) 1993, Ted A. Campbell

Copyright (c) 1995-1997, Jon B. Volkoff


COMEIG AT JULY 31 1984
ORDER =  1
Real part Frank, Imag part Moler
TAU= 0  AT ITN  1
EIGENSOLUTIONS
EIGENVALUE 1=( 1, -1)
VECTOR
( 1, 0)
RESIDUALS
( 0, 0)
ORDER =  5
Real part Frank, Imag part Moler
TAU= 56  AT ITN  1
TAU= 7.2107502  AT ITN  2
TAU= 1.6167632  AT ITN  3
TAU= 0.141359  AT ITN  4
TAU= 0.0004139  AT ITN  5
TAU= 0  AT ITN  6
EIGENSOLUTIONS
EIGENVALUE 1=( 11.438233, 3.8907870)
VECTOR
( 0.0729199, -0.3649826)
( 0.4113735, -0.2575048)
```

```
( 0.6933129, -0.1442554)
( 0.8949861, -0.0517050)
( 1.0000000, 0)
RESIDUALS
( 0, 0)
( 0, 0)
( 0, 0)
( -0, 0)
( -0, -0)
EIGENVALUE 2=( 1.957788, -0.1504239)
VECTOR
( 1, -0)
( 0.5312319, 0.4532023)
( 0.0612492, 0.4034815)
( -0.2168224, 0.1311305)
( -0.3215527, -0.0841323)
RESIDUALS
( 0, -0)
( 0, 0)
( -0, 0)
( -0, -0)
( 0, -0)
EIGENVALUE 3=( 0.916755, 0.6138752)
VECTOR
( -0.719772, -0.4457095)
( 0.5200903, -0.4259483)
( 1, 0)
( 0.2224939, 0.1771319)
( -0.7907062, 0.0761753)
RESIDUALS
( 0, 0)
( -0, 0)
( -0, 0)
( -0, -0)
( -0, -0)
EIGENVALUE 4=( 0.4072801, 0.3708246)
VECTOR
( -0.4855665, -0.2730332)
( 1, 0)
( -0.0791518, 0.1528716)
( -0.9369322, -0.0768184)
( 0.5990240, 0.0030924)
RESIDUALS
( 0, 0)
( -0, 0)
( -0, 0)
( -0, -0)
( 0, -0)
EIGENVALUE 5=( 0.2799438, 0.2749371)
VECTOR
( 0.2577608, 0.1178684)
( -0.785266, -0.0575606)
( 1, 0)
```

```
( -0.8188179, 0.0250411)
( 0.3142670, -0.0135406)
RESIDUALS
( -0, -0)
( -0, -0)
( -0, -0)
( 0, -0)
( 0, -0)
ORDER =  0
```

Pascal

Listing

```
program dr26(input,output);

{dr26.pas == eigensolutions of a complex matrix by Eberlein's
          complex Jacobi procedure

          Copyright 1988 J.C.Nash
}
{constype.def ==
  This file contains various definitions and type statements which are
  used throughout the collection of "Compact Numerical Methods".  In many
  cases not all definitions are needed, and users with very tight memory
  constraints may wish to remove some of the lines of this file when
  compiling certain programs.

  Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}
uses Dos, Crt; {Turbo Pascal 5.0 Modules}
{ 1. Interrupt, Unit, Interface, Implementation, Uses are reserved words now.}
{ 2. System,Dos,Crt are standard unit names in Turbo 5.0.}

const
  big = 1.0E+35;    {a very large number}
  Maxconst = 25;    {Maximum number of constants in data record}
  Maxobs = 100;     {Maximum number of observations in data record}
  Maxparm  = 25;    {Maximum number of parameters to adjust}
  Maxvars = 10;     {Maximum number of variables in data record}
  acctol = 0.0001;  {acceptable point tolerance for minimisation codes}
  maxm = 20;        {Maximum number or rows in a matrix}
  maxn = 20;        {Maximum number of columns in a matrix}
  maxmn = 40;       {maxn+maxm, the number of rows in a working array}
  maxsym = 210;     {maximum number of elements of a symmetric matrix
              which need to be stored = maxm * (maxm + 1)/2 }
  reltest = 10.0;   {a relative size used to check equality of numbers.
              Numbers x and y are considered equal if the
              floating-point representation of reltest+x equals
              that of reltest+y.}
  stepredn = 0.2;   {factor to reduce stepsize in line search}
```

```
  yearwrit = 1990;  {year in which file was written}

type
  str2 = string[2];
  rmatrix = array[1..maxm, 1..maxn] of real; {a real matrix}
  wmatrix = array[1..maxmn, 1..maxn] of real; {a working array, formed
                  as one real matrix stacked on another}
  smatvec = array[1..maxsym] of real; {a vector to store a symmetric matrix
              as the row-wise expansion of its lower triangle}
  rvector = array[1..maxm] of real;  {a real vector. We will use vectors
              of m elements always. While this is NOT space efficient,
              it simplifies program codes.}
  cgmethodtype= (Fletcher_Reeves,Polak_Ribiere,Beale_Sorenson);
    {three possible forms of the conjugate gradients updating formulae}
  probdata = record
          m     : integer; {number of observations}
          nvar  : integer; {number of variables}
          nconst: integer; {number of constants}
          vconst: array[1..Maxconst] of real;
          Ydata : array[1..Maxobs, 1..Maxvars] of real;
          nlls  : boolean; {true if problem is nonlinear least squares}
        end;
{
  NOTE: Pascal does not let us define the work-space for the function
  within the user-defined code.  This is a weakness of Pascal for this
  type of work.

}
var {global definitions}
  banner     : string[80]; {program name and description}

function calceps:real;
{calceps.pas ==
  This function returns the machine EPSILON or floating point tolerance,
  the smallest positive real number such that 1.0 + EPSILON > 1.0.
  EPSILON is needed to set various tolerances for different algorithms.
  While it could be entered as a constant, I prefer to calculate it, since
  users tend to move software between machines without paying attention to
  the computing environment. Note that more complete routines exist.
}
var
  e,e0: real;
  i: integer;
begin {calculate machine epsilon}
  e0 := 1; i:=0;
  repeat
    e0 := e0/2; e := 1+e0;  i := i+1;
  until (e=1.0) or (i=50); {note safety check}
  e0 := e0*2;
{ Writeln('Machine EPSILON =',e0);}
  calceps:=e0;
end; {calceps}
```

```
Procedure matrixin(var m, n: integer; var A: rmatrix;
              var avector: smatvec; var sym :boolean);

{matrixin.pas --

  This procedure generates an m by n real matrix in both
  A or avector.

  A is of type rmatrix, an array[1..nmax, 1..nmax] of real
  where nmax >= n for all possible n to be provided.

  avector is of type rvector, an array[1..nmax*(nmax+1)/2]
  of real, with nmax as above.

  sym is set true if the resulting matrix is symmetric.
}

var
  temp : real;
  i,j,k: integer;
  inchar: char;
  mtype: integer;
  mn : integer;

begin
  if (m=0) or (n=0) then
  begin
    writeln;
    writeln('******* Matrix dimensions zero *********');
    halt;
  end;
  writeln('Matrixin.pas -- generate or input a real matrix ',m,' by ',n);
  writeln('Possible matrices to generate:');
  writeln('0) Keyboard or console file input');
  writeln('1) Hilbert segment');
  writeln('2) Ding Dong');
  writeln('3) Moler');
  writeln('4) Frank symmetric');
  writeln('5) Bordered symmetric');
  writeln('6) Diagonal');
  writeln('7) Wilkinson W+');
  writeln('8) Wilkinson W-');
  writeln('9) Constant');
  writeln('10) Unit');
{ Note: others could be added.}
  mn:=n;
  if m>mn then mn:=m; {mn is maximum of m and n}
  write('Enter type to generate ');
  readln(mtype);
  writeln(mtype);
  case mtype of
    0: begin
      sym:=false;
```

```
  if m=n then
  begin
    write('Is matrix symmetric? ');  readln(inchar);
    writeln(inchar);
    if (inchar='y') or (inchar='Y') then sym:=true else sym:=false;
  end; {ask if symmetric}
  if sym then
  begin
    for i:=1 to n do
    begin
      writeln('Row ',i,' lower triangle elements');
      for j:=1 to i do
      begin
        read(A[i,j]);
        write(A[i,j]:10:5,' ');
        A[j,i]:=A[i,j];
        if (7*(j div 7) = j) and (j<i) then writeln;
      end;
      writeln;
    end;
  end {symmetric matrix}
  else
  begin {not symmetric}
    for i:=1 to m do
    begin
      writeln('Row ',i);
      for j:=1 to n do
      begin
        read(A[i,j]);
        write(A[i,j]:10:5,' ');
      end; {loop on j}
      writeln;
    end; {loop on i}
  end; {else not symmetric}
end; {case 0 -- input of matrix}
1: begin {Hilbert}
  for i:=1 to mn do
    for j:=1 to mn do
      A[i,j]:=1.0/(i+j-1.0);
  if m=n then sym:=true;
end;
2: begin {Ding Dong}
  for i:=1 to mn do
    for j:=1 to mn do
      A[i,j]:=0.5/(1.5+n-i-j);
  if m=n then sym:=true;
end;
3: begin {Moler}
  for i:=1 to mn do
  begin
    for j:=1 to i do
    begin
      A[i,j]:=j-2.0;
```

```
        A[j,i]:=j-2.0;
      end;
      A[i,i]:=i;
      if m=n then sym:=true;
    end;
  end;
  4: begin {Frank symmetric}
    for i:=1 to mn do
      for j:=1 to i do
      begin
        A[i,j]:=j;
        A[j,i]:=j;
      end;
      if m=n then sym:=true;
  end;
  5: begin {Bordered}
    temp:=2.0;
    for i:=1 to (mn-1) do
    begin
      temp:=temp/2.0; {2^(1-i)}
      for j:=1 to mn do
        A[i,j]:=0.0;
      A[i,mn]:=temp;
      A[mn,i]:=temp;
      A[i,i]:=1.0;
    end;
    A[mn,mn]:=1.0;
    if m=n then sym:=true;
  end;
  6: begin {Diagonal}
    for i:=1 to mn do
    begin
      for j:=1 to mn do
        A[i,j]:=0.0;
      A[i,i]:=i;
    end;
    if m=n then sym:=true;
  end;
  7: begin {W+}
    k:=mn div 2; {[n/2]}
    for i:=1 to mn do
      for j:=1 to mn do
        A[i,j]:=0.0;
    if m=n then sym:=true;
    for i:=1 to k do
    begin
      A[i,i]:=k+1-i;
      A[mn-i+1,mn-i+1]:=k+1-i;
    end;
    for i:=1 to mn-1 do
    begin
      A[i,i+1]:=1.0;
      A[i+1,i]:=1.0;
```

```
    end;
  end;
  8: begin {W-}
    k:=mn div 2; {[n/2]}
    for i:=1 to mn do
      for j:=1 to mn do
        A[i,j]:=0.0;
    if m=n then sym:=true;
    for i:=1 to k do
    begin
      A[i,i]:=k+1-i;
      A[mn-i+1,mn-i+1]:=i-1-k;
    end;
    for i:=1 to mn-1 do
    begin
      A[i,i+1]:=1.0;
      A[i+1,i]:=1.0;
    end;
    if m=n then sym:=true;
  end;
  9: begin {Constant}
    write('Set all elements to a constant value = ');
    readln(temp);
    writeln(temp);
    for i:=1 to mn do
      for j:=1 to mn do
        A[i,j]:=temp;
    if m=n then sym:=true;
  end;
  10: begin {Unit}
    for i:=1 to mn do
    begin
      for j:=1 to mn do A[i,j]:=0.0;
      A[i,i]:=1.0;
    end;
    if m=n then sym:=true;
  end;
  else {case statement else} {!!!! Note missing close bracket here}
  begin
    writeln;
    writeln('*** ERROR *** unrecognized option');
    halt;
  end; {else of case statement}
end; {case statement}
if sym then
begin {convert to vector form}
  k:=0; {index for vector element}
  for i:=1 to n do
  begin
    for j:=1 to i do
    begin
      k:=k+1;
      avector[k]:=A[i,j];
```

```
      end;
    end;
  end;
end; {matrixin}

procedure comeig( n : integer;
          var itcount: integer;
          var A, Z, T, U : rmatrix);

var
  Rvec : rvector;

  i, itlimit, j, k, k1, m, n1 : integer;
  aki, ami, bv, br, bi : real;
  c, c1i, c1r, c2i, c2r, ca, cb, ch, cos2a, cot2x, cotx, cx : real;
  d, de, di, diag, dr, e, ei, er, eps, eta, g, hi, hj, hr : real;
  isw, max, nc, nd, root1, root2, root : real;
  s, s1i, s1r, s2i, s2r, sa, sb, sh, sig, sin2a, sx : real;
  tanh, tau, te, tee, tem, tep ,tse, zki, zmi : real;
  mark : boolean;

begin

  writeln('alg26.pas -- comeig');
  eps := Calceps;
  mark := false; n1 := n-1;
  for i := 1 to n do
  begin
    for j := 1 to n do
    begin
      T[i,j] := 0.0; U[i,j] := 0.0; if i=j then T[i,i] := 1.0;
    end;
  end;
  itlimit := itcount;
  itcount := 0;
  while (itcount<=itlimit) and (not mark) do
  begin
    itcount := itcount+1;
    tau := 0.0;
    diag := 0.0;
    for k := 1 to n do
    begin
      tem := 0;
      for i := 1 to n do if i<>k then tem := tem+ABS(A[i,k])+ABS(Z[i,k]);
      tau := tau+tem; tep := abs(A[k,k])+abs(Z[k,k]);
      diag := diag+tep;
      Rvec[k] := tem+tep;
    end;
    writeln('TAU=',tau,'  AT ITN ',itcount);
    for k := 1 to n1 do
    begin
      max := Rvec[k]; i := k; k1 := k+1;
      for j := k1 to n do
```

```
    begin
      if max<Rvec[j] then
      begin
        max := Rvec[j]; i := j;
      end;
    end;
    if i<>k then
    begin
      Rvec[i] := Rvec[k];
      for j := 1 to n do
      begin
        tep := A[k,j]; A[k,j] := A[i,j]; A[i,j] := tep; tep := Z[k,j];
        Z[k,j] := Z[i,j]; Z[i,j] := tep;
      end;
      for j := 1 to n do
      begin
        tep := A[j,k]; A[j,k] := A[j,i]; A[j,i] := tep; tep := Z[j,k];
        Z[j,k] := Z[j,i]; Z[j,i] := tep; tep := T[j,k]; T[j,k] := T[j,i];
        T[j,i] := tep; tep := U[j,k]; U[j,k] := U[j,i]; U[j,i] := tep;
      end;
    end;
  end;
  if tau>=100.0*eps then
  begin
    mark := true;
    for k := 1 to n1 do
    begin
      k1 := k+1;
      for m := k1 to n do
      begin
        hj := 0.0; hr := 0.0; hi := 0.0; g := 0.0;
        for i := 1 to n do
        begin
          if (i<>k) and (i<>m) then
          begin
            hr := hr+A[k,i]*A[m,i]+Z[k,i]*Z[m,i];
            hr := hr-A[i,k]*A[i,m]-Z[i,k]*Z[i,m];
            hi := hi+Z[k,i]*A[m,i]-A[k,i]*Z[m,i];
            hi := hi-A[i,k]*Z[i,m]+Z[i,k]*A[i,m];
            te := A[i,k]*A[i,k]+Z[i,k]*Z[i,k]+A[m,i]*A[m,i]+Z[m,i]*Z[m,i];
            tee := A[i,m]*A[i,m]+Z[i,m]*Z[i,m]+A[k,i]*A[k,i]+Z[k,i]*Z[k,i];
            g := g+te+tee; hj := hj-te+tee;
          end;
        end;
        br := A[k,m]+A[m,k]; bi := Z[k,m]+Z[m,k]; er := A[k,m]-A[m,k];
        ei := Z[k,m]-Z[m,k]; dr := A[k,k]-A[m,m]; di := Z[k,k]-Z[m,m];
        te := br*br+ei*ei+dr*dr; tee := bi*bi+er*er+di*di;
        if te>=tee then
        begin
          isw := 1.0; c := br; s := ei; d := dr; de := di;
          root2 := sqrt(te);
        end
        else
```

```
begin
  isw := -1.0; c := bi; s := -er; d := di; de := dr;
  root2 := sqrt(tee);
end;
root1 := sqrt(s*s+c*c); sig := -1.0; if d>=0.0 then sig := 1.0;
sa := 0.0; ca := -1.0; if c>=0.0 then ca := 1.0;
if root1<=eps then
begin
  sx := 0.0; sa := 0.0; cx := 1.0; ca := 1.0;
  if isw<=0.0 then
  begin
    e := ei; bv := -br;
  end
  else
  begin
    e := er; bv := bi;
  end;
  nd := d*d+de*de;
end
else
begin
  if abs(s)>eps then
  begin
    ca := c/root1; sa := s/root1;
  end;
  cot2x := d/root1; cotx := cot2x+(sig*sqrt(1.0+cot2x*cot2x));
  sx := sig/sqrt(1.0+cotx*cotx); cx := sx*cotx;

  eta := (er*br+ei*bi)/root1; tse := (br*bi-er*ei)/root1;
  te := sig*(tse*d-de*root1)/root2; tee := (d*de+root1*tse)/root2;
  nd := root2*root2+tee*tee; tee := hj*cx*sx; cos2a := ca*ca-sa*sa;
  sin2a := 2.0*ca*sa; tem := hr*cos2a+hi*sin2a;
  tep := hi*cos2a-hr*sin2a; hr := hr*cx*cx-tem*sx*sx-ca*tee;
  hi := hi*cx*cx+tep*sx*sx-sa*tee;
  bv := isw*te*ca+eta*sa; e := ca*eta-isw*te*sa;
end;

s := hr-sig*root2*e; c := hi-sig*root2*bv; root := sqrt(c*c+s*s);
if root<eps then
begin
  cb := 1.0; ch := 1.0; sb := 0.0; sh := 0.0;
end
else
begin
  cb := -c/root; sb := s/root; tee := cb*bv-e*sb; nc := tee*tee;
  tanh := root/(g+2.0*(nc+nd)); ch := 1.0/sqrt(1.0-tanh*tanh);
  sh := ch*tanh;
end;
tem := sx*sh*(sa*cb-sb*ca); c1r := cx*ch-tem; c2r := cx*ch+tem;
c1i := -sx*sh*(ca*cb+sa*sb); c2i := c1i; tep := sx*ch*ca;
tem := cx*sh*sb; s1r := tep-tem; s2r := -tep-tem; tep := sx*ch*sa;
tem := cx*sh*cb; s1i := tep+tem; s2i := tep-tem;
tem := sqrt(s1r*s1r+s1i*s1i); tep := sqrt(s2r*s2r+s2i*s2i);
```

```
          if tep>eps then mark := false;
          if (tep>eps) and (tem>eps) then
          begin
            for i := 1 to n do
            begin
              aki := A[k,i]; ami := A[m,i]; zki := Z[k,i]; zmi := Z[m,i];
              A[k,i] := c1r*aki-c1i*zki+s1r*ami-s1i*zmi;
              Z[k,i] := c1r*zki+c1i*aki+s1r*zmi+s1i*ami;
              A[m,i] := s2r*aki-s2i*zki+c2r*ami-c2i*zmi;
              Z[m,i] := s2r*zki+s2i*aki+c2r*zmi+c2i*ami;
            end;
            for i := 1 to n do
            begin
              aki := A[i,k]; ami := A[i,m]; zki := Z[i,k]; zmi := Z[i,m];
              A[i,k] := c2r*aki-c2i*zki-s2r*ami+s2i*zmi;
              Z[i,k] := c2r*zki+c2i*aki-s2r*zmi-s2i*ami;
              A[i,m] := -s1r*aki+s1i*zki+c1r*ami-c1i*zmi;
              Z[i,m] := -s1r*zki-s1i*aki+c1r*zmi+c1i*ami;
              aki := T[i,k]; ami := T[i,m]; zki := U[i,k]; zmi := U[i,m];
              T[i,k] := c2r*aki-c2i*zki-s2r*ami+s2i*zmi;
              U[i,k] := c2r*zki+c2i*aki-s2r*zmi-s2i*ami;
              T[i,m] := -s1r*aki+s1i*zki+c1r*ami-c1i*zmi;
              U[i,m] := -s1r*zki-s1i*aki+c1r*zmi+c1i*ami;
            end;
          end;
        end;
      end;
    end
    else mark := true;
  end;
  if itcount>itlimit then itcount := -itcount;
end;

procedure stdceigv(n: integer;
                var T, U: rmatrix);

var
  i, k, m : integer;
  b, e, g, s : real;

begin
  writeln('alg11.pas -- standardized eigensolutions');
  for i := 1 to n do
  begin
    g := T[1,i]*T[1,i]+U[1,i]*U[1,i];
    k := 1;
    if n>1 then
    begin
      for m := 2 to n do
      begin
        b := T[m,i]*T[m,i]+U[m,i]*U[m,i];
        if b>g then
        begin
```

```
        k := m;
        g := b;
      end;
    end;
  end;
  e := T[k,i]/g;
  s := -U[k,i]/g;
  for k := 1 to n do
  begin
    g := T[k,i]*e-U[k,i]*s; U[k,i] := U[k,i]*e+T[k,i]*s; T[k,i] := g;
  end;
 end;
end;

procedure comres( i, n: integer;
                  A, Z, T, U, Acopy, Zcopy : rmatrix);
var
  j, k: integer;
  g, s, ss : real;

begin
  writeln('alg12.pas -- complex eigensolution residuals');
  ss := 0.0;
  for j := 1 to n do
  begin
    s := -A[i,i]*T[j,i]+Z[i,i]*U[j,i]; g := -Z[i,i]*T[j,i]-A[i,i]*U[j,i];
    for k := 1 to n do
    begin
      s := s+Acopy[j,k]*T[k,i]-Zcopy[j,k]*U[k,i];
      g := g+Acopy[j,k]*U[k,i]+Zcopy[j,k]*T[k,i];
    end;
    writeln('(',s,',',g,')');
    ss := ss+s*s+g*g;
  end;
  writeln('Sum of squares = ',ss);
end;

{Main program}
var
  A, Z, Acopy, Zcopy, T, U : rmatrix;
  i, it, j, k, n : integer;
  sym : boolean;
  avec : smatvec; {for compatibility of Matrixin only}

begin
  banner:='dr26.pas -- Eigensolutions of a general complex matrix';
  write(' Order of matrix = '); readln(n);
  writeln(n);
  writeln('Provide real part of matrix (A)');
  matrixin(n,n,A,avec,sym);
  writeln('Provide imaginary part of matrix (Z)');
  matrixin(n,n,Z,avec,sym);
  for i:=1 to n do
```

```
  begin
    for j:=1 to n do
    begin
    Acopy[i,j]:=A[i,j]; Zcopy[i,j]:=Z[i,j];
    write('(',A[i,j]:10:5,',',Z[i,j]:10:5,') ');
    if (3 * (j div 3) = j) and (j<n) then writeln;
    end; {copy loop j}
    writeln;
  end; {copy loop i}
  it:=50; {allow a maximum of 50 iterations}
  comeig( n, it, A, Z, T, U);
  if it>0 then writeln('Converged in ',it,' iterations')
    else writeln('Not converged after ',it,' iterations');
  stdceigv(n, T, U); {standardize the eigensolutions -- alg11.pas}
  for i:=1 to n do
  begin
    writeln('EIGENVALUE ',i,'=(',A[i,i],',',Z[i,i],')');
    writeln('VECTOR');
    for k:=1 to n do
    begin
    writeln('(',T[k,i],',',U[k,i],')');
    end; { loop on  k}
    comres( i, n, A, Z, T, U, Acopy, Zcopy); {residuals -- alg12.pas}
  end; {loop on i}
end. {dr26.pas == eigensolutions of a complex matrix}
```

Example output

We create an order 5 complex matrix where the real part is a Frank matrix and the imaginary part is a Moler matrix.

```
fpc ../Pascal2021/dr26.pas
# copy to run file
mv ../Pascal2021/dr26 ../Pascal2021/dr26.run
../Pascal2021/dr26.run <../Pascal2021/dr26p.in >../Pascal2021/dr26p.out
```

```
## Free Pascal Compiler version 3.0.4+dfsg-23 [2019/11/25] for x86_64
## Copyright (c) 1993-2017 by Florian Klaempfl and others
## Target OS: Linux for x86-64
## Compiling ../Pascal2021/dr26.pas
## Linking ../Pascal2021/dr26
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 594 lines compiled, 0.1 sec
```

```
 Order of matrix = 5
Provide real part of matrix (A)
Matrixin.pas -- generate or input a real matrix 5 by 5
Possible matrices to generate:
0) Keyboard or console file input
1) Hilbert segment
2) Ding Dong
3) Moler
4) Frank symmetric
5) Bordered symmetric
6) Diagonal
```

```
7) Wilkinson W+
8) Wilkinson W-
9) Constant
10) Unit
Enter type to generate 4
Provide imaginary part of matrix (Z)
Matrixin.pas -- generate or input a real matrix 5 by 5
Possible matrices to generate:
0) Keyboard or console file input
1) Hilbert segment
2) Ding Dong
3) Moler
4) Frank symmetric
5) Bordered symmetric
6) Diagonal
7) Wilkinson W+
8) Wilkinson W-
9) Constant
10) Unit
Enter type to generate 3
(   1.00000,   1.00000) (   1.00000,  -1.00000) (   1.00000,  -1.00000)
(   1.00000,  -1.00000) (   1.00000,  -1.00000)
(   1.00000,  -1.00000) (   2.00000,   2.00000) (   2.00000,   0.00000)
(   2.00000,   0.00000) (   2.00000,   0.00000)
(   1.00000,  -1.00000) (   2.00000,   0.00000) (   3.00000,   3.00000)
(   3.00000,   1.00000) (   3.00000,   1.00000)
(   1.00000,  -1.00000) (   2.00000,   0.00000) (   3.00000,   1.00000)
(   4.00000,   4.00000) (   4.00000,   2.00000)
(   1.00000,  -1.00000) (   2.00000,   0.00000) (   3.00000,   1.00000)
(   4.00000,   2.00000) (   5.00000,   5.00000)
alg26.pas -- comeig
TAU= 5.6000000000000000E+001  AT ITN 1
TAU= 7.2107501562138063E+000  AT ITN 2
TAU= 1.6167631945882213E+000  AT ITN 3
TAU= 1.4135909863720170E-001  AT ITN 4
TAU= 4.1387397594418944E-004  AT ITN 5
TAU= 2.3406515779719136E-008  AT ITN 6
TAU= 3.4129840545673970E-014  AT ITN 7
Converged in 7 iterations
alg11.pas -- standardized eigensolutions
EIGENVALUE 1=( 1.1438233058170844E+001, 5.8907869982801460E+000)
VECTOR
( 7.2919928669437584E-002,-3.6498261444044539E-001)
( 4.1137352665876853E-001,-2.5750477494283192E-001)
( 6.9331292735519889E-001,-1.4425541881377915E-001)
( 8.9498613998517162E-001,-5.1704964902128843E-002)
( 9.9999999999999989E-001, 0.0000000000000000E+000)
alg12.pas -- complex eigensolution residuals
(-9.8809849191638932E-015, 1.1102230246251565E-016)
(-6.4392935428259079E-015,-7.0637939941775585E-015)
( 1.3322676295501878E-015,-5.8841820305133297E-015)
( 6.6613381477509392E-015,-6.8833827526759706E-015)
( 6.2172489379008766E-015,-4.4408920985006262E-015)
```

```
Sum of squares =  3.7553650207708380E-028
EIGENVALUE 2=( 1.9577880276110047E+000, 1.8495761391986802E+000)
VECTOR
( 1.0000000000000000E+000,-2.7755575615628914E-017)
( 5.3123189422526218E-001, 4.5320234203420851E-001)
( 6.1249198876340880E-002, 4.0348152506126495E-001)
(-2.1682241447409578E-001, 1.3113051937238548E-001)
(-3.2155273442134791E-001,-8.4132303063020955E-002)
alg12.pas -- complex eigensolution residuals
(-7.2858385991025898E-015,-1.6098233857064770E-015)
(-1.3322676295501878E-015,-2.9698465908722937E-015)
( 5.1347814888913490E-016,-4.4408920985006262E-015)
(-2.7478019859472624E-015,-3.1086244689504383E-015)
(-6.6613381477509392E-015,-2.6645352591003757E-015)
Sum of squares =  1.5494221958391976E-028
EIGENVALUE 3=( 9.1675500804230603E-001, 2.6138751925800818E+000)
VECTOR
(-7.1977202607833091E-001,-4.4570949348429373E-001)
( 5.2009033993569398E-001,-4.2594831161408075E-001)
( 9.9999999999999989E-001,-6.9388939039072284E-018)
( 2.2249389617958162E-001, 1.7713193887438106E-001)
(-7.9070621353809867E-001, 7.6175261316525300E-002)
alg12.pas -- complex eigensolution residuals
( 3.4416913763379853E-015, 2.9976021664879227E-015)
( 1.3322676295501878E-015,-2.8865798640254070E-015)
(-3.3306690738754696E-016,-4.2188474935755949E-015)
( 2.2204460492503131E-016, 8.8817841970012523E-016)
(-7.7715611723760958E-016, 5.7731597280508140E-015)
Sum of squares =  8.3619255953427251E-029
EIGENVALUE 4=( 4.0728014995617290E-001, 2.3708245672136488E+000)
VECTOR
(-4.8556654637005797E-001,-2.7303322645238509E-001)
( 9.9999999999999989E-001, 0.0000000000000000E+000)
(-7.9151794360700342E-002, 1.5287163429309791E-001)
(-9.3693223105011536E-001,-7.6818410030158116E-002)
( 5.9902398680012370E-001, 3.0923993850483986E-003)
alg12.pas -- complex eigensolution residuals
(-5.0653925498522767E-016, 9.9920072216264089E-016)
( 0.0000000000000000E+000, 7.3552275381416621E-016)
( 2.8449465006019636E-016,-2.2204460492503131E-015)
(-7.6327832942979512E-016,-6.6613381477509392E-016)
(-7.9797279894933126E-016,-8.8817841970012523E-016)
Sum of squares =  9.2592452453819041E-030
EIGENVALUE 5=( 2.7994375621967177E-001, 2.2749371027274541E+000)
VECTOR
( 2.5776076693001215E-001, 1.1786835850188734E-001)
(-7.8526601960503983E-001,-5.7560559889801660E-002)
( 9.9999999999999989E-001, 0.0000000000000000E+000)
(-8.1881790998991266E-001, 2.5041147656711178E-002)
( 3.1426697408278631E-001,-1.3540625375941799E-002)
alg12.pas -- complex eigensolution residuals
(-2.7061686225238191E-016,-1.6653345369377348E-016)
( 1.1102230246251565E-016,-1.0824674490095276E-015)
```

```
(-6.7307270867900115E-016,-4.9960036108132044E-016)
(-6.8001160258290838E-016, 1.2212453270876722E-015)
(-1.3600232051658168E-015, 6.6613381477509392E-016)
Sum of squares =  6.2349093054620180E-030
```

Algorithm 27 – Hooke and Jeeves pattern search minimization

Pascal

Listing

```
program dr27(input,output);
{dr27.pas == driver for Hooke and Jeeves method

  This program is designed to minimise functions of n parameters.

  Present example uses the problem file ROSEN.PAS, which must be
  replaced with similar code for the user's problem.

          Copyright 1988 J.C.Nash
}
{constype.def ==
  This file contains various definitions and type statements which are
  used throughout the collection of "Compact Numerical Methods".  In many
  cases not all definitions are needed, and users with very tight memory
  constraints may wish to remove some of the lines of this file when
  compiling certain programs.

  Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}

const
  big = 1.0E+35;    {a very large number}
  Maxconst = 25;    {Maximum number of constants in data record}
  Maxobs = 100;     {Maximum number of observations in data record}
  Maxparm  = 25;    {Maximum number of parameters to adjust}
  Maxvars = 10;     {Maximum number of variables in data record}
  acctol = 0.0001;  {acceptable point tolerance for minimisation codes}
  maxm = 20;        {Maximum number or rows in a matrix}
  maxn = 20;        {Maximum number of columns in a matrix}
  maxmn = 40;       {maxn+maxm, the number of rows in a working array}
  maxsym = 210;     {maximum number of elements of a symmetric matrix
              which need to be stored = maxm * (maxm + 1)/2 }
  reltest = 10.0;   {a relative size used to check equality of numbers.
              Numbers x and y are considered equal if the
              floating-point representation of reltest+x equals
              that of reltest+y.}
  stepredn = 0.2;   {factor to reduce stepsize in line search}
  yearwrit = 1990;  {year in which file was written}

type
```

```
  str2 = string[2];
  rmatrix = array[1..maxm, 1..maxn] of real; {a real matrix}
  wmatrix = array[1..maxmn, 1..maxn] of real; {a working array, formed
                  as one real matrix stacked on another}
  smatvec = array[1..maxsym] of real; {a vector to store a symmetric matrix
              as the row-wise expansion of its lower triangle}
  rvector = array[1..maxm] of real;  {a real vector. We will use vectors
              of m elements always. While this is NOT space efficient,
              it simplifies program codes.}
  cgmethodtype= (Fletcher_Reeves,Polak_Ribiere,Beale_Sorenson);
    {three possible forms of the conjugate gradients updating formulae}
  probdata = record
          m     : integer; {number of observations}
          nvar  : integer; {number of variables}
          nconst: integer; {number of constants}
          vconst: array[1..Maxconst] of real;
          Ydata : array[1..Maxobs, 1..Maxvars] of real;
          nlls  : boolean; {true if problem is nonlinear least squares}
        end;
{
  NOTE: Pascal does not let us define the work-space for the function
  within the user-defined code.  This is a weakness of Pascal for this
  type of work.

}
var {global definitions}
  banner     : string[80]; {program name and description}

function calceps:real;
{calceps.pas ==
  This function returns the machine EPSILON or floating point tolerance,
  the smallest positive real number such that 1.0 + EPSILON > 1.0.
  EPSILON is needed to set various tolerances for different algorithms.
  While it could be entered as a constant, I prefer to calculate it, since
  users tend to move software between machines without paying attention to
  the computing environment. Note that more complete routines exist.
}
var
  e,e0: real;
  i: integer;
begin {calculate machine epsilon}
  e0 := 1; i:=0;
  repeat
    e0 := e0/2; e := 1+e0;  i := i+1;
  until (e=1.0) or (i=50); {note safety check}
  e0 := e0*2;
{ Writeln('Machine EPSILON =',e0);}
  calceps:=e0;
end; {calceps}

(* remove the comments and delete the inclusion of ROSEN.PAS
   to use the JJACF.PAS test with EX27R.CNM
   {$I JJACF.PAS}
```

```
   Note that we move the inclusion to the right just in case.
*)

{rosen.pas
  == suite of procedures and functions defining the Rosenbrock
    banana shaped valley problem.
}
procedure fminset(var n:integer;var Bvec: rvector; var Workdata: probdata);
{sets up problem and defines starting values of Bvec}
{setup for Rosenbrock problem from rosen.pas}
begin
  writeln('Function: Rosenbrock Banana Valley');
  n:=2;
  Workdata.m:=2; {for nonlinear least squares problems}
  Workdata.nvar:=0;
  Bvec[1]:=-1.2;
  Bvec[2]:=1.0;
  writeln('Classical starting point (-1.2,1)');
end; {fminset from rosen.pas}

function fminfn(n: integer; var Bvec: rvector; var Workdata:probdata;
            var nocomp:boolean):real;
{this is the Rosenbrock banana valley function from rosen.pas}
begin
  nocomp:=false; {never undefined here}
  fminfn:=sqr(Bvec[2]-sqr(Bvec[1]))*100.0+sqr(1.0-Bvec[1]);
end; {fminfn from rosen.pas}
procedure fmingr(n:integer;Bvec:rvector; var Workdata:probdata;
                                                   var g:rvector);
{computes the gradient of the Rosenbrock banana valley at point Bvec
  from rosen.pas}
begin
  g[1]:=-400.0*Bvec[1]*(Bvec[2]-sqr(Bvec[1]))-2.0*(1.0-Bvec[1]);
  g[2]:=200.0*(Bvec[2]-sqr(Bvec[1]));
end; {fmingrad from rosen.pas}

function nlres(i, n : integer; Bvec: rvector; var nocomp: boolean;
                                           var Workdata: probdata): real;
{computes residuals for the nonlinear least squares form of the
  Rosenbrock function from rosen.pas}
var
  temp: real;
begin
  nocomp:=false; {never set here}
  case i of
    1: begin
      temp:=10.0*(Bvec[2]-sqr(Bvec[1]));
    end;
    2: begin
      temp:=1.0-Bvec[1];
    end;
    else halt; {safety stop}
  end; {case}
```

```
  nlres := temp; {assign residual}
end; {nlres from rosen.pas}
procedure nljac(i, n: integer; Bvec: rvector; var jacrow: rvector;
                                              var Workdata: probdata);
{computes derivatives of residuals for the nonlinear least squares
  form of the Rosenbrock function from rosen.pas}
begin
  case i of
    1: begin
      jacrow[1]:=-20.0*Bvec[1];
      jacrow[2]:=10.0;
    end;
    2: begin
      jacrow[1]:=-1.0;
      jacrow[2]:=0.0;
    end;
    else halt; {safety stop}
  end; {case}
end; {nljac from rosen.pas}

{end of rosen.pas test function code suite}
procedure hjmin(n: integer;
        var B,X: rvector;
        var Fmin: real;
            Workdata: probdata;
        var fail: boolean;
            intol: real);

var
  i: integer;
  stepsize: real;
  fold: real;
  fval: real;
  notcomp: boolean;
  temp: real;
  samepoint: boolean;
  ifn: integer;

begin
  if intol<0.0 then intol := calceps;
  ifn := 1;
  fail := false;

  stepsize := 0.0;
  for i := 1 to n do
    if stepsize < stepredn*abs(B[i]) then stepsize := stepredn*abs(B[i]);
  if stepsize=0.0 then stepsize := stepredn;

  for i := 1 to n do X[i] := B[i];

  fval := fminfn(n, B,Workdata,notcomp);
  if notcomp then
  begin
```

```
    writeln('*** FAILURE *** Function not computable at initial point');
    fail := true;
  end
  else
  begin
    writeln('Initial function value =',fval);
    for i := 1 to n do
    begin
      write(B[i]:10:5,' ');
      if (7 * (i div 7) = i) and (i<n) then writeln;
    end;
    writeln;
    fold := fval; Fmin := fval;
    while stepsize>intol do
    begin

      for i := 1 to n do
      begin
        temp := B[i]; B[i] := temp+stepsize;
        fval := fminfn(n, B,Workdata,notcomp); ifn := ifn+1;
        if notcomp then fval := big;
        if fval<Fmin then
          Fmin := fval
        else
        begin
          B[i] := temp-stepsize;
          fval := fminfn(n, B,Workdata,notcomp); ifn := ifn+1;
          if notcomp then fval := big;
          if fval<Fmin then
            Fmin := fval
          else
            B[i] := temp;
        end;
      end;
      if Fmin<fold then
      begin

        for i := 1 to n do
        begin
          temp := 2.0*B[i]-X[i];
          X[i] := B[i]; B[i] := temp;
        end;
        fold := Fmin;
      end
      else
      begin
        samepoint := true;
        i := 1;
        repeat
          if B[i]<>X[i] then samepoint := false;
          i := i+1;
        until (not samepoint) or (i>n);
        if samepoint then
```

```
        begin
          stepsize := stepsize*stepredn;

          write('stepsize now ',stepsize:10,'  Best fn value=',Fmin);
          writeln(' after ',ifn);
          for i := 1 to n do
          begin
            write(B[i]:10:5,' ');
            if (7 * (i div 7) = i) and (i<n) then  writeln;
          end;
          writeln;
        end
        else
        begin
          for i := 1 to n do B[i] := X[i];
          writeln('Return to old base point');
        end;
      end;
    end;
    writeln('Converged to Fmin=',Fmin,' after ',ifn,' evaluations');
  end;
end;

{main program}
var
  n          : integer; {the order of the problem}
  B          : rvector; {current set of parameters}
  X          : rvector; {"best" set of parameters}
  Workdata   : probdata; { the problem data type from CONSTYPE.DEF}
  i          : integer;
  Fmin       : real;   {for the minimal function value found}
  fail       : boolean; {set TRUE if the method fails in some way}
  mytol      : real; {to store a convergence tolerance}

begin
  banner:='dr27.pas -- driver for Hooke & Jeeves minimisation';
  fminset(n,B,Workdata); {sets up problem and defines starting
                values of B}
  mytol:=-1.0; {Note: set the tolerance negative to indicate that procedure
           must obtain an appropriate value.}
  hjmin(n,B,X,Fmin,Workdata,fail,mytol); {minimise the function}
  writeln;
  writeln(' Minimum function value found =',Fmin);
  writeln(' At parameters');
  for i:=1 to n do
  begin
    writeln(' B[',i,']=',X[i]);
  end; {loop to write out parameters}
end. {dr27.pas -- Hooke & Jeeves driver}
```

Example output

Use Rosenbrock banana-shaped valley problem in 2 dimensions.

```
fpc ../Pascal2021/dr27.pas
# copy to run file
mv ../Pascal2021/dr27 ../Pascal2021/dr27.run
../Pascal2021/dr27.run  >../Pascal2021/dr27p.out
```

```
## Free Pascal Compiler version 3.0.4+dfsg-23 [2019/11/25] for x86_64
## Copyright (c) 1993-2017 by Florian Klaempfl and others
## Target OS: Linux for x86-64
## Compiling ../Pascal2021/dr27.pas
## Linking ../Pascal2021/dr27
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 303 lines compiled, 0.1 sec
```

```
Function: Rosenbrock Banana Valley
Classical starting point (-1.2,1)
Initial function value = 2.4199999999999996E+001
  -1.20000    1.00000
Return to old base point
stepsize now  4.80E-002  Best fn value= 4.4562559999999998E+000 after 12
  -0.96000    1.00000
Return to old base point
stepsize now  9.60E-003  Best fn value= 4.0578692096000006E+000 after 24
  -1.00800    1.00000
Return to old base point
Return to old base point
Return to old base point
stepsize now  1.92E-003  Best fn value= 1.0707319193525812E-003 after 161
   0.98880    0.98080
Return to old base point
stepsize now  3.84E-004  Best fn value= 1.3884319308353293E-004 after 172
   0.99072    0.98080
Return to old base point
stepsize now  7.68E-005  Best fn value= 9.3512661164573335E-005 after 184
   0.99034    0.98080
Return to old base point
stepsize now  1.54E-005  Best fn value= 1.1312841503153136E-007 after 387
   0.99978    0.99954
Return to old base point
stepsize now  3.07E-006  Best fn value= 5.6836523263813657E-008 after 399
   0.99977    0.99954
Return to old base point
stepsize now  6.14E-007  Best fn value= 5.2964452813167824E-008 after 410
   0.99977    0.99954
Return to old base point
stepsize now  1.23E-007  Best fn value= 1.4270706145809712E-011 after 506
   1.00000    0.99999
Return to old base point
stepsize now  2.46E-008  Best fn value= 9.4796228739601164E-012 after 517
   1.00000    0.99999
Return to old base point
stepsize now  4.92E-009  Best fn value= 9.4690619892340589E-012 after 529
   1.00000    0.99999
Return to old base point
Return to old base point
```

```
stepsize now  9.83E-010  Best fn value= 4.4567065650768205E-015 after 661
   1.00000    1.00000
Return to old base point
stepsize now  1.97E-010  Best fn value= 4.0727302462025689E-015 after 673
   1.00000    1.00000
Return to old base point
stepsize now  3.93E-011  Best fn value= 5.5957365459244406E-019 after 1084
   1.00000    1.00000
Return to old base point
stepsize now  7.86E-012  Best fn value= 1.7697158812184515E-019 after 1096
   1.00000    1.00000
Return to old base point
stepsize now  1.57E-012  Best fn value= 1.5428882521329409E-019 after 1107
   1.00000    1.00000
Return to old base point
stepsize now  3.15E-013  Best fn value= 2.9714142551459652E-023 after 1190
   1.00000    1.00000
Return to old base point
stepsize now  6.29E-014  Best fn value= 3.6918757417520296E-024 after 1201
   1.00000    1.00000
Return to old base point
stepsize now  1.26E-014  Best fn value= 2.5495050765906063E-024 after 1213
   1.00000    1.00000
Return to old base point
stepsize now  2.52E-015  Best fn value= 4.7610344079933319E-027 after 1293
   1.00000    1.00000
Return to old base point
stepsize now  5.03E-016  Best fn value= 3.9955928108960689E-027 after 1304
   1.00000    1.00000
Converged to Fmin= 3.9955928108960689E-027 after 1304 evaluations

 Minimum function value found = 3.9955928108960689E-027
 At parameters
 B[1]= 9.9999999999993683E-001
 B[2]= 9.9999999999987343E-001
```

Cleanup of working files

The following script is included to remove files created during compilation or execution of the examples.

```
## remove object and run files
cd ../fortran/
echo `pwd`
rm  *.o
rm  *.run
# rm *.out
cd ../Pascal2021/
echo `pwd`
rm  *.o
rm  *.run
# rm *.out
cd ../BASIC
echo `pwd`
# rm *.out
```

```
cd ../Documentation
## ?? others
```

```
## /j19z/j19store/versioned/Nash-Compact-Numerical-Methods/fortran
## rm: cannot remove '*.o': No such file or directory
## /j19z/j19store/versioned/Nash-Compact-Numerical-Methods/Pascal2021
## /j19z/j19store/versioned/Nash-Compact-Numerical-Methods/BASIC
```

References

Chartres, B. A. 1962. "Adaptation of the Jacobi Method for a Computer with Magnetic-tape Backing Store." *The Computer Journal* 5 (1): 51–60.

Forsythe, G. E., and P. Henrici. 1960. "The Cyclic Jacobi Method for Computing the Principal Values of a Complex Matrix." *Trans. Amer. Math. Soc.* 94: 1–23.

Hestenes, Magnus R. 1958. "Inversion of Matrices by Biorthogonalization and Related Results." *Journal of the Society for Industrial and Applied Mathematics* 6 (1): 51–90. http://www.jstor.org/stable/2098862.

Kaiser, H. F. 1972. "The JK Method: A Procedure for Finding the Eigenvectors and Eigenvalues of a Real Symmetric Matrix." *Computer Journal* 15 (3): 271–73.

Nash, John C. 1975. "A One-Sided Transformation Method for the Singular Decomposition and Algebraic Eigenproblem." *Computer Journal* 18 (1): 74–76.

———. 1979. *Compact Numerical Methods for Computers : Linear Algebra and Function Minimisation.* Book. Hilger: Bristol.

Nash, John C., and Seymour Shlien. 1987. "Simple Algorithms for the Partial Singular Value Decomposition." *Computer Journal* 30 (3): 268–75.

Wilkinson, J. H., C. Reinsch, and F. L. Bauer. 1971. *Linear Algebra.* Die Grundlehren Der Mathematischen Wissenschaften in Einzeldarstellungen, v. 10. Springer-Verlag.