Algorithms in the Nashlib set in various programming languages

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

11/01/2021

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 some comments and/or details relating to the algorithms themselves or their implementations.

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
      ND1=ND+1
   1  READ(NIN,900)M,NVAR
 900  FORMAT(10I4)
      WRITE(NOUT,950)M,NVAR
 950  FORMAT(' TEST USING FRANK MATRIX',I4,' BY',I4)
      IF(M.LE.0.OR.NVAR.LE.0)STOP
      CALL A3PREP(M,NVAR,D,MD,FRANKM)
      WRITE(NOUT,952)
 952  FORMAT(' D MATRIX')
      CALL OUT(D,MD,M,NVAR,NOUT)
  11  READ(NIN,900)IPOS
      WRITE(NOUT,951)IPOS
 951  FORMAT(' COL. #S OF INDEPENDENT VARIABLES'/10I4)
      N=0
  20  N=N+1
      IF(IPOS(N).LE.0)GOTO 30
      K=IPOS(N)
      DO 25 J=1,M
        A(J,N)=D(J,K)
  25  CONTINUE
      GOTO 20
  30  N=N-1
      IF(N.EQ.0)GOTO 1
      ESVD=.FALSE.
      WRITE(NOUT,953)
 953  FORMAT(' A MATRIX')
      CALL OUT(A,MD,M,N,NOUT)
  35  READ(NIN,900)YPOS
      WRITE(NOUT,954)YPOS
 954  FORMAT(' DEPENDENT VARIABLE = COL.',I4)
      IF(YPOS.LE.0)GOTO 11
      DO 40 I=1,M
        Y(I)=D(I,YPOS)
  40  CONTINUE
      WRITE(NOUT,955)(Y(I),I=1,M)
```

```
 955  FORMAT(1H ,5E16.8)
      NTOL=.FALSE.
  50  READ(NIN,902)Q
 902  FORMAT(E16.8)
      WRITE(NOUT,956)Q
 956  FORMAT(' SING. VALS. .LE.',E16.8,'  ARE PRESUMED ZERO')
      IF(Q.LT.0.0)GOTO 35
C  IBM MACHINE PRECISION VALUE
      EPS=16.0**(-5)
      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)
      GOTO 50
      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
```

```fortran
  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

```
## #!/bin/bash
gfortran ../fortran/dr0102.f
./a.out < ../fortran/dr0102.txt
```

```
##   TEST USING FRANK MATRIX   4 BY   5
##   D MATRIX
##   ROW  1
##     1.00000000E+00  1.00000000E+00  1.00000000E+00  1.00000000E+00  4.00000000E+00
##   ROW  2
##     1.00000000E+00  2.00000000E+00  2.00000000E+00  2.00000000E+00  7.00000000E+00
##   ROW  3
##     1.00000000E+00  2.00000000E+00  3.00000000E+00  3.00000000E+00  9.00000000E+00
##   ROW  4
##     1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  1.00000000E+01
##   COL. #S OF INDEPENDENT VARIABLES
##     1   2   0   0   0   0   0   0   0   0
##   A MATRIX
##   ROW  1
##     1.00000000E+00  1.00000000E+00
##   ROW  2
##     1.00000000E+00  2.00000000E+00
##   ROW  3
##     1.00000000E+00  2.00000000E+00
##   ROW  4
##     1.00000000E+00  2.00000000E+00
##   DEPENDENT VARIABLE = COL.   3
##     0.10000000E+01  0.20000000E+01  0.30000000E+01  0.30000000E+01
##   SING. VALS. .LE.  0.00000000E+00  ARE PRESUMED ZERO
##      1 ROTATIONS
##      1 ROTATIONS
##      0 ROTATIONS
##   SV(  1)=  4.10142136E+00
##   SV(  2)=  4.22304988E-01
##   SINGULAR VALUE(  1)=  4.10142136E+00
##   SINGULAR VALUE(  2)=  4.22304988E-01
##   X(  1)= -6.66668236E-01
##   X(  2)=  1.66666770E+00
##   SING. VALS. .LE. -0.10000000E+01  ARE PRESUMED ZERO
##   DEPENDENT VARIABLE = COL.   4
##     0.10000000E+01  0.20000000E+01  0.30000000E+01  0.40000000E+01
##   SING. VALS. .LE.  0.00000000E+00  ARE PRESUMED ZERO
##      0 ROTATIONS
##   SV(  1)=  1.00000000E+00
##   SV(  2)=  1.00000000E+00
##   SINGULAR VALUE(  1)=  1.00000000E+00
##   SINGULAR VALUE(  2)=  1.00000000E+00
##   X(  1)=  5.23438358E+00
##   X(  2)=  7.75390148E-01
##   SING. VALS. .LE. -0.10000000E+01  ARE PRESUMED ZERO
##   DEPENDENT VARIABLE = COL.   5
##     0.40000000E+01  0.70000000E+01  0.90000000E+01  0.10000000E+02
##   SING. VALS. .LE.  0.00000000E+00  ARE PRESUMED ZERO
##      0 ROTATIONS
##   SV(  1)=  1.00000000E+00
```

```
## SV(  2)=  1.00000000E+00
## SINGULAR VALUE(  1)=  1.00000000E+00
## SINGULAR VALUE(  2)=  1.00000000E+00
## X(  1)=  1.54891491E+01
## X(  2)=  1.19147384E+00
## SING. VALS. .LE. -0.10000000E+01  ARE PRESUMED ZERO
## DEPENDENT VARIABLE = COL.  -1
## COL. #S OF INDEPENDENT VARIABLES
##    1   2   3   4   5   0   0   0   0   0
## A MATRIX
## ROW  1
##    1.00000000E+00  1.00000000E+00  1.00000000E+00  1.00000000E+00  4.00000000E+00
## ROW  2
##    1.00000000E+00  2.00000000E+00  2.00000000E+00  2.00000000E+00  7.00000000E+00
## ROW  3
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  3.00000000E+00  9.00000000E+00
## ROW  4
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  1.00000000E+01
## DEPENDENT VARIABLE = COL.   1
##    0.10000000E+01  0.10000000E+01  0.10000000E+01  0.10000000E+01
## SING. VALS. .LE.  0.00000000E+00  ARE PRESUMED ZERO
##    10 ROTATIONS
##    10 ROTATIONS
##    10 ROTATIONS
##     7 ROTATIONS
##     2 ROTATIONS
##     0 ROTATIONS
## SV(  1)=  1.77387428E+01
## SV(  2)=  1.03549778E+00
## SV(  3)=  4.29354817E-01
## SV(  4)=  2.83528328E-01
## SV(  5)=  2.69620699E-07
## SINGULAR VALUE(  1)=  1.77387428E+01
## SINGULAR VALUE(  2)=  1.03549778E+00
## SINGULAR VALUE(  3)=  4.29354817E-01
## SINGULAR VALUE(  4)=  2.83528328E-01
## SINGULAR VALUE(  5)=  2.69620699E-07
## X(  1)= -1.02215825E+06
## X(  2)= -1.02215925E+06
## X(  3)= -1.02216019E+06
## X(  4)= -1.02215994E+06
## X(  5)=  1.02215969E+06
## SING. VALS. .LE.  0.99999997E-04  ARE PRESUMED ZERO
##     5 ROTATIONS
##     8 ROTATIONS
##     3 ROTATIONS
##     0 ROTATIONS
## SV(  1)=  1.41421354E+00
## SV(  2)=  9.99999940E-01
## SV(  3)=  9.99999940E-01
## SV(  4)=  9.99999881E-01
## SV(  5)=  5.12674944E-07
## SINGULAR VALUE(  1)=  1.41421354E+00
## SINGULAR VALUE(  2)=  9.99999940E-01
```

```
##  SINGULAR VALUE(  3)=  9.99999940E-01
##  SINGULAR VALUE(  4)=  9.99999881E-01
##  SINGULAR VALUE(  5)=  5.12674944E-07
##  X(  1)=  1.90189278E+00
##  X(  2)=  2.30421573E-01
##  X(  3)= -1.78813830E-01
##  X(  4)=  1.13637932E-01
##  X(  5)= -3.08124572E-01
##  SING. VALS. .LE. -0.10000000E+01  ARE PRESUMED ZERO
##  DEPENDENT VARIABLE = COL.  -1
##  COL. #S OF INDEPENDENT VARIABLES
##   -1   0   0   0   0   0   0   0   0   0
##  TEST USING FRANK MATRIX  20 BY  10
##  D MATRIX
##  ROW  1
##    1.00000000E+00  1.00000000E+00  1.00000000E+00  1.00000000E+00  1.00000000E+00
##    1.00000000E+00  1.00000000E+00  1.00000000E+00  1.00000000E+00  1.05254097E-35
##  ROW  2
##    1.00000000E+00  2.00000000E+00  2.00000000E+00  2.00000000E+00  2.00000000E+00
##    2.00000000E+00  2.00000000E+00  2.00000000E+00  2.00000000E+00  4.59135442E-41
##  ROW  3
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  3.00000000E+00  3.00000000E+00
##    3.00000000E+00  3.00000000E+00  3.00000000E+00  3.00000000E+00  1.05254068E-35
##  ROW  4
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  4.00000000E+00
##    4.00000000E+00  4.00000000E+00  4.00000000E+00  4.00000000E+00  4.59135442E-41
##  ROW  5
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    5.00000000E+00  5.00000000E+00  5.00000000E+00  5.00000000E+00  1.98118902E+38
##  ROW  6
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  6.00000000E+00  6.00000000E+00  6.00000000E+00  4.57005468E-41
##  ROW  7
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  7.00000000E+00  7.00000000E+00  1.32515691E+38
##  ROW  8
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  8.00000000E+00  4.57005468E-41
##  ROW  9
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00  0.00000000E+00
##  ROW 10
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00  0.00000000E+00
##  ROW 11
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00  1.13645305E-42
##  ROW 12
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00  0.00000000E+00
##  ROW 13
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00  1.31939589E+38
##  ROW 14
```

```
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00  4.57005468E-41
##  ROW 15
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00  1.55032756E+38
##  ROW 16
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00  4.57005468E-41
##  ROW 17
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00  3.28687239E-22
##  ROW 18
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00  0.00000000E+00
##  ROW 19
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00  1.02033236E-38
##  ROW 20
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00  0.00000000E+00
##  COL. #S OF INDEPENDENT VARIABLES
##    1   2   3   0   0   0   0   0   0   0
##  A 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
##  ROW  6
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW  7
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW  8
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW  9
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW 10
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW 11
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW 12
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW 13
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW 14
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW 15
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW 16
```

```
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW 17
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW 18
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW 19
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  ROW 20
##    1.00000000E+00  2.00000000E+00  3.00000000E+00
##  DEPENDENT VARIABLE = COL.   4
##    0.10000000E+01  0.20000000E+01  0.30000000E+01  0.40000000E+01  0.40000000E+01
##    0.40000000E+01  0.40000000E+01  0.40000000E+01  0.40000000E+01  0.40000000E+01
##    0.40000000E+01  0.40000000E+01  0.40000000E+01  0.40000000E+01  0.40000000E+01
##    0.40000000E+01  0.40000000E+01  0.40000000E+01  0.40000000E+01  0.40000000E+01
##  SING. VALS. .LE.  0.00000000E+00  ARE PRESUMED ZERO
##     3 ROTATIONS
##     3 ROTATIONS
##     2 ROTATIONS
##     2 ROTATIONS
##     0 ROTATIONS
##  SV(  1)=  1.62246857E+01
##  SV(  2)=  8.09399605E-01
##  SV(  3)=  3.23070347E-01
##  SINGULAR VALUE(  1)=  1.62246857E+01
##  SINGULAR VALUE(  2)=  8.09399605E-01
##  SINGULAR VALUE(  3)=  3.23070347E-01
##  X(  1)=  1.60932541E-06
##  X(  2)= -9.44444716E-01
##  X(  3)=  1.94444454E+00
##  SING. VALS. .LE.  0.99999997E-04  ARE PRESUMED ZERO
##     1 ROTATIONS
##     0 ROTATIONS
##  SV(  1)=  1.00000012E+00
##  SV(  2)=  1.00000000E+00
##  SV(  3)=  9.99999702E-01
##  SINGULAR VALUE(  1)=  1.00000012E+00
##  SINGULAR VALUE(  2)=  1.00000000E+00
##  SINGULAR VALUE(  3)=  9.99999702E-01
##  X(  1)=  1.68351212E+01
##  X(  2)= -1.22364354E+00
##  X(  3)= -3.69865030E-01
##  SING. VALS. .LE. -0.10000000E+01  ARE PRESUMED ZERO
##  DEPENDENT VARIABLE = COL.  -1
##  COL. #S OF INDEPENDENT VARIABLES
##   -1   0   0   0   0   0   0   0   0   0
##  TEST USING FRANK MATRIX   0 BY   0
## Note: The following floating-point exceptions are signalling: IEEE_DENORMAL
```

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
# now execute it
../Pascal2021/dr0102 > ../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
```

```r
              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) {
```

```r
        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
```

```r
# 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
```

```r
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
```

```r
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
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
```

```
# 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?

==

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
./a.out < ../fortran/a3data.txt > ../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

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
      DO 100 N=2,10,2
      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)
 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
./a.out
```

```
## 0ORDER=   2  ORIGINAL MATRIX
##  ROW  1
##    1.00000000E+00
##  ROW  2
##    1.00000000E+00  2.00000000E+00
## 0INVERSE
##  ROW  1
##    2.00000000E+00
##  ROW  2
##   -1.00000000E+00  1.00000000E+00
## 0INVERSE OF INVERSE
##  ROW  1
##    1.00000000E+00
##  ROW  2
##    1.00000000E+00  2.00000000E+00
## 0MAX. DEVN. OF INVERSE-INVERSE FROM ORIGINAL=  0.00000000E+00
## 0ORDER=   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
## 0INVERSE
##  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
## OORDER=   6  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
##  ROW  5
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##  ROW  6
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+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  2.00000000E+00
##  ROW  5
##    0.00000000E+00  0.00000000E+00  0.00000000E+00 -1.00000000E+00  2.00000000E+00
##  ROW  6
##    0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00 -1.00000000E+00
##    1.00000000E+00
## OINVERSE OF INVERSE
##  ROW  1
##    1.00000000E+00
##  ROW  2
##    1.00000000E+00  2.00000000E+00
##  ROW  3
##    1.00000012E+00  2.00000024E+00  3.00000048E+00
##  ROW  4
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  3.99999976E+00
##  ROW  5
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  3.99999952E+00  4.99999952E+00
##  ROW  6
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  3.99999976E+00  4.99999952E+00
##    5.99999952E+00
## OMAX. DEVN. OF INVERSE-INVERSE FROM ORIGINAL=  4.76837158E-07
## OORDER=   8  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
## ROW 5
##   1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
## ROW 6
##   1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##   6.00000000E+00
## ROW 7
##   1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##   6.00000000E+00  7.00000000E+00
## ROW 8
##   1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##   6.00000000E+00  7.00000000E+00  8.00000000E+00
## 0INVERSE
## 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  2.00000000E+00
## ROW 5
##   0.00000000E+00  0.00000000E+00  0.00000000E+00 -1.00000000E+00  2.00000000E+00
## ROW 6
##   0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00 -1.00000000E+00
##   2.00000000E+00
## ROW 7
##   0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00
##  -1.00000000E+00  2.00000000E+00
## ROW 8
##   0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00
##   0.00000000E+00 -1.00000000E+00  1.00000000E+00
## 0INVERSE OF INVERSE
## ROW 1
##   1.00000000E+00
## ROW 2
##   9.99999881E-01  1.99999976E+00
## ROW 3
##   9.99999940E-01  1.99999988E+00  2.99999952E+00
## ROW 4
##   9.99999702E-01  1.99999940E+00  2.99999928E+00  3.99999857E+00
## ROW 5
##   9.99999642E-01  1.99999928E+00  2.99999905E+00  3.99999809E+00  4.99999762E+00
## ROW 6
##   9.99999523E-01  1.99999905E+00  2.99999857E+00  3.99999762E+00  4.99999714E+00
##   5.99999666E+00
## ROW 7
```

```
##    9.99999523E-01  1.99999905E+00  2.99999857E+00  3.99999762E+00  4.99999714E+00
##    5.99999619E+00  6.99999619E+00
##  ROW  8
##    9.99999523E-01  1.99999905E+00  2.99999857E+00  3.99999762E+00  4.99999714E+00
##    5.99999619E+00  6.99999619E+00  7.99999619E+00
## OMAX. DEVN. OF INVERSE-INVERSE FROM ORIGINAL=  3.81469727E-06
## OORDER=  10  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
##  ROW  5
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##  ROW  6
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00
##  ROW  7
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00
##  ROW  8
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00
##  ROW  9
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00
##  ROW 10
##    1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
##    6.00000000E+00  7.00000000E+00  8.00000000E+00  9.00000000E+00  1.00000000E+01
## 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  2.00000000E+00
##  ROW  5
##    0.00000000E+00  0.00000000E+00  0.00000000E+00 -1.00000000E+00  2.00000000E+00
##  ROW  6
##    0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00 -1.00000000E+00
##    2.00000000E+00
##  ROW  7
##    0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00
##   -1.00000000E+00  2.00000000E+00
##  ROW  8
##    0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00
##    0.00000000E+00 -1.00000000E+00  2.00000000E+00
##  ROW  9
##    0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00
```

```
##    0.00000000E+00  0.00000000E+00 -1.00000000E+00  2.00000000E+00
##  ROW 10
##    0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00
##    0.00000000E+00  0.00000000E+00  0.00000000E+00 -1.00000000E+00  1.00000000E+00
## OINVERSE OF INVERSE
##  ROW  1
##    1.00000012E+00
##  ROW  2
##    1.00000000E+00  2.00000000E+00
##  ROW  3
##    1.00000012E+00  2.00000024E+00  2.99999976E+00
##  ROW  4
##    9.99999821E-01  1.99999964E+00  2.99999976E+00  3.99999905E+00
##  ROW  5
##    9.99999881E-01  1.99999976E+00  2.99999952E+00  3.99999905E+00  4.99999905E+00
##  ROW  6
##    9.99999762E-01  1.99999952E+00  2.99999952E+00  3.99999833E+00  4.99999809E+00
##    5.99999762E+00
##  ROW  7
##    9.99999762E-01  1.99999952E+00  2.99999928E+00  3.99999857E+00  4.99999809E+00
##    5.99999762E+00  6.99999714E+00
##  ROW  8
##    9.99999821E-01  1.99999964E+00  2.99999952E+00  3.99999857E+00  4.99999857E+00
##    5.99999809E+00  6.99999809E+00  7.99999857E+00
##  ROW  9
##    9.99999762E-01  1.99999952E+00  2.99999952E+00  3.99999857E+00  4.99999857E+00
##    5.99999762E+00  6.99999762E+00  7.99999809E+00  8.99999809E+00
##  ROW 10
##    9.99999762E-01  1.99999952E+00  2.99999952E+00  3.99999857E+00  4.99999857E+00
##    5.99999762E+00  6.99999762E+00  7.99999809E+00  8.99999809E+00  9.99999809E+00
## OMAX. DEVN. OF INVERSE-INVERSE FROM ORIGINAL=  2.86102295E-06
```

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

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/dr09.pas
../Pascal2021/dr09 >../Pascal2021/dr09.out
```

```
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)
```

```python
# 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

Others

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

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

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.