Algorithms in the Nashlib set in various programming languages – Part 1

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

Note that these are single precision codes. 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.

Listing – Algorithm 1

```
      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
```

Listing – Algorithm 2

```
      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"
6 REM Some issues are because Data General BASIC only
7 REM allowed A-Z variable names!
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
200 REM RATIONALIZE STORAGE??
210 DIM Y(M,N+1),A(M,N),T(N,N),G(N),X(N),Z(N),U(N),B(M)
236 REM PRINT "Prep matrix and RHS"
237 REM This is a VERY pedestrian way to do this.
239 PRINT "Matrix A:"
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
520 LET A(I,J)=Y(I,J)
525 PRINT A(I,J);
530 NEXT J
535 PRINT
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);
1400 NEXT J
1405 PRINT
1410 REM U in A, V in T, S in Z
1420 PRINT "U"
1422 FOR I=1 TO M
1424 FOR J=1 TO N
1426   PRINT A(I,J);
1428 NEXT J
1430 PRINT
1432 NEXT I
1440 PRINT "V"
1442 FOR I=1 TO M
1444 FOR J=1 TO N
1446   PRINT A(I,J);
1448 NEXT J
1450 PRINT
1450 NEXT I
1600 LET Q=1E-20
1605 REM UT*B
1610 PRINT "SINGULAR VALUES <=",Q," ARE TAKEN AS 0"
1615 REM PRINT "G:";
1620 FOR J=1 TO N
1630 LET S=0
1640 FOR I=1 TO M
1650  LET S=S+A(I,J)*B(I)
1652 NEXT I
1655 IF (Z(J) > Q) THEN LET G(J)=S/Z(J) ELSE G(J)=0
1666 REM PRINT G(J);
1670 NEXT J
1671 REM PRINT
1680 PRINT "Least squares solution"
1690 FOR I=1 TO N
1700 LET S=0
1710 FOR J=1 TO N
```

```
1720 LET S=S+T(I,J)*G(J)
1730 NEXT J
1740 LET X(I)=S
1745 PRINT S;
1750 NEXT I
1760 PRINT
1800 quit
```

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
## Matrix A:
##  5 0.0000010 1
##  6 0.9999990 1
##  7 2.0000100 1
##  8 2.9999000 1
##
## SWEEP         1
## SWEEP         2
##
## CONVERGENCE AT SWEEP        3
##
## SINGULAR VALUES
##  13.7529874 1.6896078 0.0000119
## U
##  0.358943 -0.7557625 -0.3286873
##  0.4465265 -0.3171936 0.1117406
##  0.53411 0.1213826 0.7626745
##  0.6216916 0.5598907 -0.5457163
## V
##  0.358943 -0.7557625 -0.3286873
##  0.4465265 -0.3171936 0.1117406
##  0.53411 0.1213826 0.7626745
##  0.6216916 0.5598907 -0.5457163
## SINGULAR VALUES <=          0              ARE TAKEN AS 0
## Least squares solution
##  0.9999975 0.0000025 -3.9999876
##
## done
```

Pascal

Listing – Algorithm 1

```
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;
```

Listing – Algorithm 2

```
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;
```

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.

```r
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

```
# 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
```

```
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
```

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

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

```r
# 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
```

```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.

For solving the least squares problem, 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.

Notes on work in 2021 with Sidey Timmons

Starting in late 2019, one of us (JN) exchanged ideas relating to the one-sided Jacobi approach to the singular value decomposition. The application was to work to estimate the magnetic field of Jupiter from spacecraft measurements. The compact Nash codes were attractive as extremely high precision calculations were needed that required special programming language processors, for example, FM Fortran (Smith (1991), see also http://dmsmith.lmu.build/).

As this work progressed, it was noted that work on LAPACK had, in 2000, revealed quite large discrepancies between some implementations of the Givens' method for the QR decomposition, and a working paper was prepared, eventually published as Bindel et al. (2002). Some aspects of these ideas will be included in this collection. (This comment written as of 2021-3-21.)

Fortran

Listing – Algorithm 3

```
      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
```

Example output

The following output presents an example using the Frank matrix as a test. 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/dr03.f
mv ./a.out ../fortran/dr03.run
../fortran/dr03.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 – Algorithm 3, column-wise approach

```
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;
```

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
```

R

Listing – Algorithm 3, column-wise approach

```
a3gr <- function(A){
    # Givens decomosition A to Q R where R is in original A
    m <- dim(A)[1]
    n <- dim(A)[2]
    Q <- diag(m)
    tol <- .Machine$double.eps^(1.5)
#  ?? should we use 2 or 1.5 or ??
    mn <- min(m, n)
#  STEP 1
    if (m == 1) {
      sol <- list(Q=Q, Rx=A)
      return(sol)
    }
    for (j in 1:mn){
       if (j+1 > m) break # end loop
#  C  STEP 2
      for (k in (j+1):m){
         # C  STEP 3
         C <- A[j,j]
         S <- A[k,j]
         B <- abs(C)
         if (abs(S) > B) {B <- abs(S)}
         if (B == 0.0) break # 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) <= tol) {break} # GOTO 90 Note: <= rather than <
#    C  STEP 6
         C <- C/P
#    C  STEP 7
         for (i in 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
         for (i in 1:m){
            P <- Q[i,j]
            Q[i,j] <- C*P+S*Q[i,k]
            Q[i,k] <- -S*P+C*Q[i,k]
         }
# C  STEP 9   90    CONTINUE
      }
# C  STEP 10   100  CONTINUE
    }
    sol <- list(Q=Q, Rx=A)
    sol
}
## C  TEST ALGORITHM 3
# m <- as.numeric(readline("no. of rows="))
# n <- as.numeric(readline("no of columns="))
m <- 5
n <- 3
# build frank matrix
cat("Frank matrix A ",m," by ",n,"\n")
A <- matrix(0.0, nrow=m, ncol=n)
for (i in 1:m){
    for (j in 1:n){
        A[i,j] <- min(i,j)-2.0
    }
    if (i <= j) {A[i,i] <- i}
}
print(A)
Acopy <- A
grA <- a3gr(A)
Q <- grA$Q
cat("Q\n")
print(Q)
R <- grA$Rx
cat("R\n")
print(R)
test <- Q %*% R
cat("error =",max(abs(test-Acopy)),"\n")
```

Example output

Below is a first try that uses explicit loops that are known to be inefficient in R. In this version, we work across the columns in the outer loop. For simplicity in running the code within `knitr`, the input of matrix dimensions has been replaced with simple assignments.

```
a3gr <- function(A){
    # Givens decomosition A to Q R where R is in original A
    m <- dim(A)[1]
    n <- dim(A)[2]
    Q <- diag(m)
```

```
    tol <- .Machine$double.eps^(1.5)
#  ?? should we use 2 or 1.5 or ??
    mn <- min(m, n)
#  STEP 1
    if (m == 1) {
      sol <- list(Q=Q, Rx=A)
      return(sol)
    }
    for (j in 1:mn){
       if (j+1 > m) break # end loop
#  C  STEP 2
      for (k in (j+1):m){
         # C  STEP 3
         C <- A[j,j]
         S <- A[k,j]
         B <- abs(C)
         if (abs(S) > B) {B <- abs(S)}
         if (B == 0.0) break # 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) <= tol) {break} # GOTO 90 Note: <= rather than <
#    C  STEP 6
         C <- C/P
#    C  STEP 7
         for (i in 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
         for (i in 1:m){
            P <- Q[i,j]
            Q[i,j] <- C*P+S*Q[i,k]
            Q[i,k] <- -S*P+C*Q[i,k]
         }
# C  STEP 9   90    CONTINUE
      }
# C  STEP 10   100  CONTINUE
    }
    sol <- list(Q=Q, Rx=A)
    sol
}
## C  TEST ALGORITHM 3
# m <- as.numeric(readline("no. of rows="))
# n <- as.numeric(readline("no of columns="))
m <- 5
n <- 3
# build frank matrix
cat("Frank matrix A ",m," by ",n,"\n")
```

```
## Frank matrix A  5  by  3
```

```
A <- matrix(0.0, nrow=m, ncol=n)
for (i in 1:m){
    for (j in 1:n){
        A[i,j] <- min(i,j)-2.0
    }
    if (i <= j) {A[i,i] <- i}
}
print(A)
```

```
##      [,1] [,2] [,3]
## [1,]    1   -1   -1
## [2,]   -1    2    0
## [3,]   -1    0    3
## [4,]   -1    0    1
## [5,]   -1    0    1
```

```
Acopy <- A
grA <- a3gr(A)
Q <- grA$Q
cat("Q\n")
```

```
## Q
```

```
print(Q)
```

```
##            [,1]       [,2]        [,3]      [,4]        [,5]
## [1,]  0.4472136 -0.2236068 -0.07624929 0.8528029  0.13005122
## [2,] -0.4472136  0.7826238 -0.03812464 0.4264014  0.06502561
## [3,] -0.4472136 -0.3354102  0.80061749 0.2132007  0.03251280
## [4,] -0.4472136 -0.3354102 -0.41937107 0.2132007 -0.68276889
## [5,] -0.4472136 -0.3354102 -0.41937107 0.0000000  0.71528170
```

```
R <- grA$Rx
cat("R\n")
```

```
## R
```

```
print(R)
```

```
##               [,1]      [,2]          [,3]
## [1,]  2.236068e+00 -1.341641 -2.683282e+00
## [2,]  1.297602e-16  1.788854 -1.453444e+00
## [3,] -3.644543e-18  0.000000  1.639360e+00
## [4,] -6.598481e-17  0.000000 -1.110223e-16
## [5,]  5.871041e-17  0.000000  0.000000e+00
```

```
test <- Q %*% R
cat("error =",max(abs(test-Acopy)),"\n")
```

```
## error = 1.332268e-15
```

We can simplify the loops for steps 7 and 8 as follows without change in the results.

```
#    C  STEP 7
         Pv <- A[j,]
         A[j,] <- C*Pv+S*A[k,]
         A[k,] <- -S*Pv+C*A[k,]
# C  STEP 8
```

```
        Pv <- Q[,j]
        Q[,j] <- C*Pv+S*Q[,k]
        Q[,k] <- -S*Pv+C*Q[,k]
```

Issues with calculating the Givens' plane rotations

Bindel et al. (2002) point out `"the LAPACK [Anderson et al. 1999] routines SLARTG, CLARTG, SLARGV and CLARGV, the Level 1 BLAS routines SROTG and CROTG [Lawson et al. 1979], as well as Algorithm 5.1.5 in Golub and Van Loan [1996] can get significantly different answers for mathematically identical inputs."`

The essence of the Givens' approach to generating a QR decomposition of a matrix is to apply a plane rotation to two rows of the current working matrix W. Let us call these rows W_j and W_k where $k > j$. Since the rest of the full plane rotation is simply an identity, we only need consider the effects on the two rows. That is, we will look at what we get from

$$\begin{Bmatrix} newW_j \\ newW_k \end{Bmatrix} = \begin{Bmatrix} c & s \\ -s & c \end{Bmatrix} \begin{Bmatrix} W_j \\ W_k \end{Bmatrix}$$

where we want the j element $newW_{k,j}$ to be zero.

Based on the LAPACK routine `slartg.f`, the following version of Algorithm 3 incorporates the suggestions of Bindel et al. (2002).

```
C&&& A3
C  TEST ALGORITHM 3
C  J.C. NASH   JULY 1978, APRIL 1989
      LOGICAL SAVEQ
      CHARACTER QSAVE
      INTEGER M,N,NIN,NOUT
      DOUBLE PRECISION 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,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
C-------------------------------------
      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
      DOUBLE PRECISION 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
C-------------------------------------
      SUBROUTINE OUT(A,NDIM,N,NP,NOUT)
C  J.C. NASH   JULY 1978, APRIL 1989
      INTEGER NDIM,N,NOUT,I,J
      DOUBLE PRECISION 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
C-------------------------------------
      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
      DOUBLE PRECISION 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
C===========================================
      SUBROUTINE A3GR(M,N,A,NDIM,Q,SAVEQ)
C Amended variant to use ideas of @Bindel2002Givens
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,COUNT
      DOUBLE PRECISION A(NDIM,N),Q(NDIM,M),TOL,B,P,S,C,F,G,F1,G1,RC
      DOUBLE PRECISION ONE, TWO, ZERO
      PARAMETER       ( ONE = 1.0D+0, ZERO = 0.0D+0, TWO = 2.0D+0 )
      DOUBLE PRECISION SAFMIN, EPS, SAFEPS, SAFMN2, SAFMX2, LSE
      DOUBLE PRECISION LL2, LL22, ASFMN2
      SAFMIN = 2.2250738585072014D-308 ! DLAMCH( 'S' )
      EPS = TWO**(-53) ! DLAMCH( 'E' )
      SAFEPS=SAFMIN/EPS
      LSE=LOG(SAFEPS)
      LL2=LSE/LOG(TWO)
      LL22=LL2/TWO
      SAFMN2 = TWO**INT(LL22)
      SAFMX2 = ONE / SAFMN2

      write(*,*)"SAFMN2=",SAFMN2,"  SAFMX2=",SAFMX2

!      ROTCHK = 0.05 ! used for inconsequential rotation test. Was 0.25
      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
           G=A(K,J)
           F=A(J,J)
           IF( G.EQ.ZERO ) THEN
               C = ONE
               S = ZERO
               RC = F
           ELSE IF( F.EQ.ZERO ) THEN
               C = ZERO
               S = ONE
               RC = G
           ELSE
               F1 = F
               G1 = G
               SCALE = MAX( ABS( F1 ), ABS( G1 ) )
               IF( SCALE.GE.SAFMX2 ) THEN
               COUNT = 0
510          CONTINUE
               COUNT = COUNT + 1
               F1 = F1*SAFMN2
               G1 = G1*SAFMN2
               SCALE = MAX( ABS( F1 ), ABS( G1 ) )
               IF( SCALE.GE.SAFMX2 .AND. COUNT .LT. 20) GO TO 510
               RC = SQRT( F1**2+G1**2 )
               C = F1 / RC
               S = G1 / RC
               DO II = 1,COUNT
                 RC = RC*SAFMX2
 520           END DO
             ELSE IF( SCALE.LE.SAFMN2 ) THEN
               COUNT = 0
 530           CONTINUE
               COUNT = COUNT + 1
               F1 = F1*SAFMX2
               G1 = G1*SAFMX2
               SCALE = MAX( ABS( F1 ), ABS( G1 ) )
               IF( SCALE.LE.SAFMN2 ) GO TO 530
               RC = SQRT( F1**2+G1**2 )
               C = F1 / RC
               S = G1 / RC
               DO II = 1, COUNT
                 RC = RC*SAFMN2
               END DO
             ELSE
```

```
              RC = SQRT( F1**2+G1**2 )
              C = F1 / RC
              S = G1 / RC
            END IF
            IF( ABS( F ).GT.ABS( G ) .AND. C.LT.ZERO ) THEN
              C = -C
              S = -S
              RC = -RC
            END IF
          END IF
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 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)
          END DO
C   STEP 9
  90    CONTINUE
C   STEP 10
 100  CONTINUE
      RETURN
      END
```

Examples of effect of Bindel et al. Givens' plane rotations

?? need some illustrative cases that are not too big.

Algorithm 4 – Row-oriented SVD and least squares solution

The essence of Algorithm 4 is to use a QR decomposition to triangularize a "long skinny" matrix, that is, m by n with $m >> n$. The resulting n by n matrix is then further decomposed to a singular value decomposition.

$$A = QR$$

$$R = PSV^T$$

Thus

$$A = QPSV^T$$

so

$$U = QP$$

There are many possibilities for both the QR and SVD parts of this process. In the early 1970s, one of us (JN) needed to solve a problem with $m = 196$ by $n = 25$ columns (variables plus a constant).

Fortran

Listing – SVD and least squares solution

We include the plane-rotation sub-program.

```
      SUBROUTINE A4LSGS(W,ND1,N1,NG,N,H,G,X,Z,IPR,NOBS,EPS,QTOL,NTOL)
C  ALGORITHM 4  LEAST SQUARES SOLUTION BY GIVENS REDUCTION  AND ROW
C     ORTHOGONALISATION SINGULAR VALUE DECOMPOSITION
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C   W IS WORKING ARRAY  N1 BY NG   DIMENSIONED ND1 BY NG
C   G  RIGHT HAND SIDES
C   N  INDEPENDENT VARIABLES (INCLUDING CONSTANT)
C   N1=N+1
C   NG=N+G
C   IPR  =  PRINT CHANNEL   IPR.GT.0 FOR PRINTING
C   NOBS =  NUMBER OF OBSERVATIONS - OUTPUT - COUNTED DURING EXECUTION
C    X   =  SOLUTION VECTOR
C    H   =  RESIDUAL SUM OF SQUARES ACCUMULATOR
C   EPS  =  MACHINE PRECISION
C  QTOL  = TOLERANCE FOR SINGULAR VALUES
C        SING. VALS. .LE. QTOL ARE TAKEN AS ZERO
C  NTOL  =  .TRUE. IF ONLY NEW VALUE OF QTOL
C  STEP 0
      LOGICAL DEND,NTOL
      REAL QTOL
      INTEGER N1,NG,N,G,IPR,NOBS,T,K,M,NM1,J1,ND1
      REAL W(ND1,NG),H(G),X(N),Z(N),EPS,TOL,S,C,P,B,Q,R
      IF(N.NE.N1-1.OR.NG.NE.N+G)STOP
      IF(NTOL)GOTO 240
      TOL=N*N*EPS*EPS
      NOBS=0
      DO 4 I=1,N
        DO 2 J=1,NG
          W(I,J)=0.0
   2    CONTINUE
   4  CONTINUE
      T=NG
      K=N1
      IF(G.LT.1)GOTO 9
      DO 6 J=1,G
        H(J)=0.0
   6  CONTINUE
C  STEP 1
   9  DEND=.FALSE.
  10  CALL INW(W,ND1,N1,NG,NOBS,DEND)
C  STEP 2
      IF(DEND)GOTO 110
C  STEP 3
      NOBS=NOBS+1
C  STEP 4
      DO 90 J=1,N
C  STEP 5
        M=J
        S=W(K,J)
```

```
        C=W(J,J)
        B=ABS(C)
        IF(ABS(S).GT.B)B=ABS(S)
C  STEP 6
        IF(B.EQ.0.0)GOTO 90
        C=C/B
        S=S/B
      P=SQRT(C**2+S**2)
C  STEP 7
        S=S/P
        IF(ABS(S).LT.TOL)GOTO 90
C  STEP 8
        C=C/P
        CALL ROTN(J,K,S,C,M,T,W,ND1,NG)
C  STEP 9
  90  CONTINUE
C  STEP 10
      IF(G.LT.1)GOTO 10
      DO 105 J=1,G
        M=N+J
        H(J)=H(J)+W(N1,M)**2
 105  CONTINUE
      GOTO 10
C  STEP 11
 110  M=1
C  STEP 12
      NM1=N-1
 120  COUNT=N*(N-1)/2
C  STEP 13
      DO 215 J=1,NM1
        J1=J+1
C  STEP 14
        DO 210 K=J1,N
C  STEP 15
          P=0.0
          Q=0.0
          R=0.0
          DO 155 I=1,N
            P=P+W(J,I)*W(K,I)
            Q=Q+W(J,I)**2
            R=R+W(K,I)**2
 155      CONTINUE
C  STEP 16
          IF(Q.GE.R)GOTO 170
          C=0.0
          S=1.0
          GOTO 190
 170      IF(Q*R.EQ.0.0)GOTO 200
C  STEP 17
          IF((P*P)/(Q*R).LT.TOL)GOTO 200
C  STEP 18
          Q=Q-R
          R=SQRT(4.0*P**2+Q**2)
```

```
          C=SQRT((R+Q)/(2.0*R))
          S=P/(R*C)
C  STEP 19
 190      CALL ROTN(J,K,S,C,M,T,W,ND1,NG)
          GOTO 210
 200      COUNT=COUNT-1
C  STEP 20
C  STEP 21
 210    CONTINUE
 215  CONTINUE
C  STEP 22
      IF(COUNT.GT.0)GOTO 120
C  STEP 23
      DO 238 J=1,N
        S=0.0
        DO 232 I=1,N
          S=S+W(J,I)**2
 232    CONTINUE
        S=SQRT(S)
        Z(J)=S
        IF(S.LT.TOL)GOTO 238
      DO 236 I=1,N
          W(J,I)=W(J,I)/S
 236    CONTINUE
 238  CONTINUE
      IF(IPR.GT.0)WRITE(IPR,983)(J,Z(J),J=1,N)
 983  FORMAT(11H SING. VAL.,I3,3H = ,1PE16.8)
C  STEP 24
 240  Q=QTOL
      IF(G.LT.1)RETURN
C  STEP 25
      DO 300 I=1,G
C  STEP 25A
        RSS=H(I)
C  STEP 26
      DO 290 J=1,N
C  STEP 27
          P=0.0
          J1=N+I
          DO 275 K=1,N
            IF(Z(K).LE.Q)GOTO 275
            P=P+W(K,J)*W(K,J1)/Z(K)
 275      CONTINUE
C  STEP 28
          X(J)=P
C         IF(Z(J).LE.Q)H(I)=H(I)+W(J,J1)**2
C  STEP 28A
          IF(Z(J).LE.Q)RSS=RSS+W(J,J1)**2
C  STEP 29
 290    CONTINUE
        IF(IPR.GT.0)WRITE(IPR,981)I,RSS
 981    FORMAT('ORESIDUAL SUM OF SQUARES FOR SOLN',I4,'=',1PE16.8)
        IF(IPR.GT.0)WRITE(IPR,982)(J,X(J),J=1,N)
```

```
 982    FORMAT( 3H X(,I3,2H)=,1PE16.8)
C  STEP 30
 300  CONTINUE
      RETURN
        END
      SUBROUTINE ROTN(J,K,S,C,M,T,W,N1,NG)
C  PLANE ROTATION  ALGORITHM 4A   J.C.NASH  JULY 1978
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
      INTEGER J,K,M,T,N1,NG,I
      REAL  S,C,W(N1,NG),R
C  STEP 1
      DO 10 I=M,T
        R=W(J,I)
        W(J,I)=R*C+S*W(K,I)
        W(K,I)=-R*S+C*W(K,I)
  10  CONTINUE
C  STEP 2
      RETURN
      END
```

```
gfortran ../fortran/dr04.f
mv ./a.out ../fortran/dr04.run
../fortran/dr04.run < ../fortran/dr04f.in > ../fortran/dr04f.out
```

```
 TESTS USING DATA 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 VARIABLES FROM COL.    3    4    5   -1    0    0    0    0    0    0
 DEP. VAR.    1 FROM COL.    3
   0.10000000E+01  0.20000000E+01  0.30000000E+01  0.30000000E+01
 DEP. VAR.    2 FROM COL.    4
   0.10000000E+01  0.20000000E+01  0.30000000E+01  0.40000000E+01
 DEP. VAR.    3 FROM COL.    5
   0.40000000E+01  0.70000000E+01  0.90000000E+01  0.10000000E+02
 SING. VALS. .LE.  0.00000000E+00  ARE PRESUMED ZERO
 SING. VAL.   1 =    4.10142136E+00
```

```
SING. VAL.  2 =   4.22305048E-01
RESIDUAL SUM OF SQUARES FOR SOLN   1=  6.66666567E-01
X(  1)= -6.66666567E-01
X(  2)=  1.66666663E+00
RESIDUAL SUM OF SQUARES FOR SOLN   2=  1.99999964E+00
X(  1)= -1.00000024E+00
X(  2)=  2.00000024E+00
RESIDUAL SUM OF SQUARES FOR SOLN   3=  4.66666555E+00
X(  1)= -6.66666627E-01
X(  2)=  4.66666651E+00
SING. VALS. .LE. -0.10000000E+01  ARE PRESUMED ZERO
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 VARIABLES FROM COL.   1   0   0   0   0   0   0   0   0   0
DEP. VAR.   1 FROM COL.   1
  0.10000000E+01  0.10000000E+01  0.10000000E+01  0.10000000E+01
SING. VALS. .LE.  0.00000000E+00  ARE PRESUMED ZERO
SING. VAL.  1 =   1.77387428E+01
SING. VAL.  2 =   1.03549814E+00
SING. VAL.  3 =   4.29354757E-01
SING. VAL.  4 =   2.83528417E-01
SING. VAL.  5 =   0.00000000E+00
RESIDUAL SUM OF SQUARES FOR SOLN   1=  0.00000000E+00
X(  1)=  7.99999416E-01
X(  2)= -2.00000614E-01
X(  3)= -2.00000435E-01
X(  4)= -2.00000376E-01
X(  5)=  1.99998289E-01
SING. VALS. .LE.  0.99999997E-04  ARE PRESUMED ZERO
RESIDUAL SUM OF SQUARES FOR SOLN   1=  0.00000000E+00
X(  1)=  7.99999416E-01
X(  2)= -2.00000614E-01
X(  3)= -2.00000435E-01
X(  4)= -2.00000376E-01
X(  5)=  1.99998289E-01
SING. VALS. .LE. -0.10000000E+01  ARE PRESUMED ZERO
COL. #S OF INDEPENDENT VARIABLES
 -1   0   0   0   0   0   0   0   0   0
TESTS USING DATA 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  0.00000000E+00
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  0.00000000E+00
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  0.00000000E+00
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  0.00000000E+00
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  0.00000000E+00
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  0.00000000E+00
ROW  7
  1.00000000E+00  2.00000000E+00  3.00000000E+00  4.00000000E+00  5.00000000E+00
  6.00000000E+00  7.00000000E+00  7.00000000E+00  7.00000000E+00  0.00000000E+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  8.00000000E+00  0.00000000E+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  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  0.00000000E+00
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  0.00000000E+00
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  0.00000000E+00
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  0.00000000E+00
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  0.00000000E+00
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  0.00000000E+00
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  0.00000000E+00
```

```
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 VARIABLES FROM COL.   4   0   0   0   0   0   0   0   0   0
DEP. VAR.   1 FROM 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
```

```
SING. VAL.  1 =   1.62246876E+01
SING. VAL.  2 =   8.09399545E-01
SING. VAL.  3 =   3.23070377E-01
RESIDUAL SUM OF SQUARES FOR SOLN   1=  9.44444954E-01
X(  1)=  1.49011612E-06
X(  2)= -9.44447219E-01
X(  3)=  1.94444609E+00
SING. VALS. .LE.  0.99999997E-04  ARE PRESUMED ZERO
RESIDUAL SUM OF SQUARES FOR SOLN   1=  9.44444954E-01
X(  1)=  1.49011612E-06
X(  2)= -9.44447219E-01
X(  3)=  1.94444609E+00
SING. VALS. .LE. -0.10000000E+01  ARE PRESUMED ZERO
COL. #S OF INDEPENDENT VARIABLES
 -1   0   0   0   0   0   0   0   0   0
TESTS USING DATA MATRIX   0 BY   0
```

Example output

Algorithms 5 and 6 – Gaussian elimination and back-solution

Fortran

Listing – Algorithm 5 Gaussian elimination

```
      SUBROUTINE A5GE(A,NA,N,NP,D,TOL)
C    ALGORITHM 5
C   J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C    GAUSS ELIMINATION WITH PARTIAL PIVOTING
C      A=WORKING ARRAY -- COLUMNS 1 TO N HAVE COEFFICIENT MATRIX
C                      -- COLUMNS N+1 TO NP HAVE RIGHT HAND SIDES
C      NA=FIRST DIMENSION OF A .GE. N
C      N=ORDER OF EQUATIONS
C      NP=N + NO. OF RIGHT HAND SIDES
C      D=DETERMINANT OF COEFFICIENT MATRIX (OUTPUT ONLY)
C      TOL=TOLERANCE FOR ZERO SCALED TO SIZE OF COEFFICINT ELEMENTS
C         TOL IS SET NEGATIVE IF COEFFICIENT MATRIX COMPUTATIONALLY
C         SINGULAR. NEGATIVE TOL ON INPUT STOPS EXECUTION
C   STEP 0
      INTEGER N,NA,NP,I,N1,J,H,K,J1
      REAL D,TOL,A(NA,NP),S
      D=1.0
      IF(TOL.LE.0.0)STOP
C   STEP 1
      N1=N-1
      DO 60 J=1,N1
C   STEP 2A
        S=ABS(A(J,J))
        K=J
C   STEP 2B
        J1=J+1
        DO 10 H=J1,N
          IF(ABS(A(H,J)).LE.S)GOTO 10
          S=ABS(A(H,J))
          K=H
  10    CONTINUE
        IF(K.EQ.J)GOTO 30
C   STEP 3
        DO 20 I=J,NP
          S=A(K,I)
          A(K,I)=A(J,I)
          A(J,I)=S
  20    CONTINUE
        D=-D
C   STEP 4
  30    D=D*A(J,J)
        IF(ABS(A(J,J)).LE.TOL)GOTO 70
C   STEP 5
        DO 50 K=J1,N
          A(K,J)=A(K,J)/A(J,J)
          DO 40 I=J1,NP
            A(K,I)=A(K,I)-A(K,J)*A(J,I)
  40      CONTINUE
```

```
  50    CONTINUE
C  STEP 6
  60  CONTINUE
C  STEP 7
      D=D*A(N,N)
      IF(ABS(A(N,N)).LE.TOL)GOTO 70
      RETURN
C  COMPUTATIONALLY SINGULAR COEFFICIENT MATRIX -- EXIT WITH TOL.LT.0.0
  70  TOL=-1.0
      RETURN
      END
```

Listing – Algorithm 6 Back-solution of upper triangular equation systems

```
      SUBROUTINE A6BS(A,NA,N,NP)
C  ALGORITHM 6
C  J.C. NASH   JULY 1978, APRIL 1989
C  BACK-SUBSTITUTION TO FOLLOW GAUSS ELIMINATION
C     A=WORKING ARRAY AS OUTPUT BY A5GE - ALGORITHM 5
C     NA=FIRST DIMENSION OF A
C     NP=N + NO. OF (TRANSFORMED) RIGHT HAND SIDES
C     N=ORDER OF EQUATIONS
C  STEP 0
      INTEGER N,NA,NP,N1,I,J,JJ,K
      REAL A(NA,NP),S
C  STEP 1
      N1=N+1
      DO 200 I=N1,NP
C  STEP 2
        A(N,I)=A(N,I)/A(N,N)
        IF(N.EQ.1)GOTO 200
C  STEP 3 - NOTE FORM FOR LOOPING
        DO 180 JJ=2,N
          J=N1-JJ
C  STEP 4
          S=A(J,I)
C  STEP 5
          J1=J+1
          DO 160 K=J1,N
            S=S-A(J,K)*A(K,I)
 160      CONTINUE
C  STEP 6
          A(J,I)=S/A(J,J)
C  STEP 7
 180    CONTINUE
C  STEP 8
 200  CONTINUE
      RETURN
      END
```

Example output

```
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 – Algorithm 5, column-wise approach

```
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;
```

Listing – Algorithm 6, back-solution of upper triangular equations

```
procedure gebacksub(n, p:integer;
                     var A : rmatrix);
var
  s : real;
  i, j, k: integer;

begin
  writeln('alg06.pas -- Gauss elimination back-substitution');
  for i:=(n+1) to (n+p) do
  begin
    A[n,i]:=A[n,i]/A[n,n];
    for j:=(n-1) downto 1 do
    begin
      s:=A[j,i];
      for k:=(j+1) to n do
      begin
        s:=s-A[j,k]*A[k,i];
      end;
      A[j,i]:=s/A[j,j];
    end;
  end;
end;
```

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 back-solution

Fortran

Listing – Algorithm 7 Choleski decomposition

```
      SUBROUTINE A7CH(A,N2,N,INDEF)
C   ALGORITHM 7
C   J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C   CHOLESKI DECOMPOSITION OF REAL-SYMMETRIC
      LOGICAL INDEF
      INTEGER N2,N,I,J,Q,M,K,J1,MK,QK
      REAL A(N2),S
      INDEF=.FALSE.
C   STEP 1
      DO 100 J=1,N
C   STEP 2
        Q=J*(J+1)/2
C   STEP 3
        IF(J.EQ.1)GOTO 50
C   STEP 4
        DO 40 I=J,N
          M=I*(I-1)/2+J
          S=A(M)
          J1=J-1
          DO 20 K=1,J1
            MK=M-K
            QK=Q-K
```

```
            S=S-A(MK)*A(QK)
  20      CONTINUE
          A(M)=S
  40    CONTINUE
C  STEP 5
  50    IF(A(Q).GT.0.0)GOTO 60
C  SET FLAG IN THIS CASE
        INDEF=.TRUE.
C  STEP 6
        A(Q)=0.0
C  ASSUMES MATRIX NON-NEGATIVE DEFINITE
C  STEP 7
  60    S=SQRT(A(Q))
C  STEP 8
        DO 80 I=J,N
          M=I*(I-1)/2+J
          IF(S.EQ.0.0)A(M)=0.0
          IF(S.GT.0.0)A(M)=A(M)/S
  80    CONTINUE
C  STEP 9
 100  CONTINUE
      RETURN
      END
```

Listing – Algorithm 8 Choleski Back-solution

```
      SUBROUTINE A8CS(A,N2,X,N)
C  ALGORITHM 8
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  CHOLESKI BACK-SOLUTION - ALGORITHM 8
C  STEP 0
      INTEGER N2,N,Q,I,I1,J,II,QJ
      REAL A(N2),X(N)
C  STEP 1
C  SAFETY CHECK ON N2
      IF(N2.NE.N*(N+1)/2)STOP
      IF(A(1).EQ.0.0)X(1)=0.0
      IF(A(1).GT.0.0)X(1)=X(1)/A(1)
C  STEP 2
      IF(N.EQ.1)GOTO 50
C  STEP 3
      Q=1
C  STEP 4
      DO 40 I=2,N
C  STEP 5
        I1=I-1
        DO 10 J=1,I1
          Q=Q+1
          X(I)=X(I)-A(Q)*X(J)
  10    CONTINUE
C  STEP 6
        Q=Q+1
C  STEP 7
```

```
        IF(A(Q).EQ.0.0)X(I)=0.0
        IF(A(Q).GT.0.0)X(I)=X(I)/A(Q)
C   STEP 8
  40  CONTINUE
C   STEP 9
  50  IF(A(N2).EQ.0.0)X(N)=0.0
      IF(A(N2).GT.0.0)X(N)=X(N)/A(N2)
C   STEP 10
      IF(N.EQ.1)GOTO 100
C   STEP 11
      DO 80 II=2,N
        I=N+2-II
C   STEP 12
        Q=I*(I-1)/2
C   STEP 13
        I1=I-1
        DO 60 J=1,I1
          QJ=Q+J
          X(J)=X(J)-X(I)*A(QJ)
  60    CONTINUE
C   STEP 14
        IF(A(Q).EQ.0.0)X(I1)=0.0
        IF(A(Q).GT.0.0)X(I1)=X(I1)/A(Q)
C   STEP 15
  80  CONTINUE
C   STEP 16
 100  RETURN
      END
```

Pascal

Listing – Algorithm 7 Choleski decomposition

```
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;
```

Listing – Algorithm 8 Choleski Back-solution

```
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

```
      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
```

```
## 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
## 0INVERSE 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
## 0MAX. 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

```
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 -*-
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
#  alg09.py
# =========================================================================
    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)
```

Example output

```
python3 ../python/dr09py.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:")
```

```r
#      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
```

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

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

```r
print(vinv)
```

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

```r
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
```

```r
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

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

Bindel, D., J. Demmel, W. Kahan, and O. Marques. 2002. "On Computing Givens Rotations Reliably and Efficiently." *ACM Transactions on Mathematical Software* 28 (2): 206–38.

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.

Smith, David M. 1991. “A Fortran Package for Floating-Point Multiple-Precision Arithmetic.” *ACM Transactions on Mathematical Software* 17: 273–83.

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