Algorithms in the Nashlib set in various programming languages – Part 3

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

11/01/2021

Contents

Abstract **2**

Overview of this document **3**

Algorithm 16 – Grid search **4**
Fortran 4
Pascal 7

Algorithm 17 – Minimize a function of one parameter **8**
Fortran 9
Pascal 12

Algorithm 18 – Roots of a function of one parameter **16**
Fortran 17
BASIC 21
Pascal 23

Algorithm 19 and 20 – Nelder-Mead minimization and Axial search **27**
Fortran 27

Algorithm 21 – Variable metric minimization **34**
Fortran 34
Pascal 37

Algorithm 22 – Conjugate gradients minimizers **42**
Fortran 42
BASIC 45
Pascal 45
R 50

Algorithm 23 – Marquardt method for nonlinear least squares **50**
Fortran 50
BASIC 53
Pascal 53

Algorithm 27 – Hooke and Jeeves pattern search minimization **58**
BASIC 58
Pascal 61
R 65

References **69**

Abstract

Algorithms 16-23 from the book Nash (1979) are implemented in a variety of programming languages including Fortran, BASIC, Pascal, Python and R. These concern rootfinding, function minimisation and nonlinear least squares.

Overview of this document

This section is repeated for each of the parts of Nashlib documentation.

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.

Algorithm 16 – Grid search

Grid search – establishing a regular pattern of parameter values for one or more arguments of a function and then evaluating that function on the “grid” – is a brute force approach to finding roots, minima, maxima and other features of a function surface. While it cannot be recommended as an efficient method for finding roots or minima, it offers a way to generate data for plotting the function surface and for localizing roots or minima when these are not unique. Furthermore, it is readily understood, and offers a useful starting point in presenting and understanding a problem.

Fortran

Listing

```
      SUBROUTINE A16GS(U,V,N,FNS,IFN,TOL,IPR,T,VAL)
C  ALGORITHM 16  GRID SEARCH
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  U,V DEFINE THE INTERVAL OF INTEREST
C  N  GIVES THE NUMBER OF DIVISIONS (N+1) POINTS
C  FNS IS THE NAME OF THE FUNCTION    VAL=FNS(B,NOCOM)
C     NOCOM SET .TRUE. IF NOT COMPUTABLE. PROGRAM HALTS IN THIS CASE
C  IFN  IS LIMIT ON FUNCTION EVALUATIONS ALLOWED. RETURNS ACTUAL USED
C  TOL =CONVERGENCE TOLERANCE ON ABS(V-U)*2/N
C  IPR = PRINT CHANNEL   IPR.GT.0 FOR PRINTING.
C  T  = LOWEST VALUE FOUND
C  VAL    =  WORKING VECTOR OF VALUES AT GRID POINTS
C  STEP 0
      LOGICAL NOCOM
      INTEGER N,K,J,LIM,N1
      REAL H,U,V,T,TOL,P,SV,X,VAL(N)
C  N.LE.2  CAN'T REDUCE INTERVAL
      IF(N.LT.3)STOP
      LIM=IFN
      IFN=0
      NOCOM = .FALSE.
      T=FNS(U,NOCOM)
      IF(NOCOM)STOP
      IFN=IFN+1
      IF(IPR.GT.0)WRITE(IPR,956)IFN,U,T
      VAL(1)=T
      SV=FNS(V,NOCOM)
      IF(NOCOM)STOP
      IFN=IFN+1
      IF(IPR.GT.0)WRITE(IPR,956)IFN,V,SV
C  STEP 1
  10  K=0
      IF(SV.GE.T)GOTO 15
      K=N
      T=SV
  15  H=(V-U)/N
C  STEP 2
C  S(U) ALREAD IN T
      N1=N-1
      DO 60 J=1,N1
C  STEP 3
```

```
        X=U+J*H
        P=FNS(X,NOCOM)
        IF(NOCOM)STOP
        IFN=IFN+1
        IF(IFN.GE.LIM)RETURN
      IF(IPR.GT.0)WRITE(IPR,956)IFN,X,P
 956  FORMAT( 8H EVALN #,I4,4H  F(,1PE16.8,2H)=,E16.8)
C   SAVE VALUE
      VAL(J+1)=P
C   STEP 4
        IF(P.GE.T)GOTO 60
C   STEP 5
        T=P
        K=J
C   STEP 6
  60  CONTINUE
C   STEP 7
      IF(ABS(H).LT.0.5*TOL)RETURN
C   STEP 8
      V=U+(K+1)*H
      U=V-2*H
      IF(K.EQ.0)GOTO 82
C   S(U) IS IN VAL(K)
      T=VAL(K)
      GOTO 84
  82  T=FNS(U,NOCOM)
      IF(NOCOM)STOP
      IFN=IFN+1
      IF(IPR.GT.0)WRITE(IPR,956)IFN,U,T
      IF(IFN.GE.LIM)RETURN
  84  IF(K.GT.N-2)GOTO 86
      SV=VAL(K+2)
      GOTO 10
  86  IF(K.EQ.N1)GOTO 10
C   SV  ALREADY IN PLACE IF K=N-1
      SV=FNS(V,NOCOM)
      IF(NOCOM)STOP
      IFN=IFN+1
      IF(IPR.GT.0)WRITE(IPR,956)IFN,V,SV
      IF(IFN.GE.LIM)RETURN
      GOTO 10
      END
```

Example output

```
gfortran ../fortran/dr16.f
mv ./a.out ../fortran/dr16f.run
../fortran/dr16f.run < ../fortran/dr16f.in
```

```
## OTEST- COUNT=   80 NBIS=    5 U=        0.00000 V=        3.00000 TOL=   0.0000100000
##  EVALN #   1  F(  0.00000000E+00)= -5.00000000E+00
##  EVALN #   2  F(  3.00000000E+00)=  1.60000000E+01
##  EVALN #   3  F(  6.00000024E-01)= -5.98400021E+00
##  EVALN #   4  F(  1.20000005E+00)= -5.67199993E+00
```

```
## EVALN #   5  F(  1.80000007E+00)= -2.76799941E+00
## EVALN #   6  F(  2.40000010E+00)=  4.02400112E+00
## EVALN #   7  F(  2.40000010E-01)= -5.46617603E+00
## EVALN #   8  F(  4.80000019E-01)= -5.84940815E+00
## EVALN #   9  F(  7.20000029E-01)= -6.06675196E+00
## EVALN #  10  F(  9.60000038E-01)= -6.03526402E+00
## EVALN #  11  F(  5.76000035E-01)= -5.96089697E+00
## EVALN #  12  F(  6.72000051E-01)= -6.04053545E+00
## EVALN #  13  F(  7.68000007E-01)= -6.08301544E+00
## EVALN #  14  F(  8.64000022E-01)= -6.08302736E+00
## EVALN #  15  F(  8.06400001E-01)= -6.08841324E+00
## EVALN #  16  F(  8.44799995E-01)= -6.08667755E+00
## EVALN #  17  F(  8.83200049E-01)= -6.07746649E+00
## EVALN #  18  F(  9.21600044E-01)= -6.06044197E+00
## EVALN #  19  F(  7.83360004E-01)= -6.08600903E+00
## EVALN #  20  F(  7.98720002E-01)= -6.08789349E+00
## EVALN #  21  F(  8.14080000E-01)= -6.08864784E+00
## EVALN #  22  F(  8.29439998E-01)= -6.08824968E+00
## EVALN #  23  F(  8.04863989E-01)= -6.08833218E+00
## EVALN #  24  F(  8.11007977E-01)= -6.08858871E+00
## EVALN #  25  F(  8.17152023E-01)= -6.08866119E+00
## EVALN #  26  F(  8.23296010E-01)= -6.08854866E+00
## EVALN #  27  F(  8.13465655E-01)= -6.08863974E+00
## EVALN #  28  F(  8.15923214E-01)= -6.08866119E+00
## EVALN #  29  F(  8.18380833E-01)= -6.08865356E+00
## EVALN #  30  F(  8.20838392E-01)= -6.08861589E+00
## EVALN #  31  F(  8.14448714E-01)= -6.08865166E+00
## EVALN #  32  F(  8.15431714E-01)= -6.08865929E+00
## EVALN #  33  F(  8.16414773E-01)= -6.08866215E+00
## EVALN #  34  F(  8.17397773E-01)= -6.08866024E+00
## EVALN #  35  F(  8.15824926E-01)= -6.08866119E+00
## EVALN #  36  F(  8.16218138E-01)= -6.08866215E+00
## EVALN #  37  F(  8.16611350E-01)= -6.08866215E+00
## EVALN #  38  F(  8.17004561E-01)= -6.08866119E+00
## EVALN #  39  F(  8.15982223E-01)= -6.08866119E+00
## EVALN #  40  F(  8.16139519E-01)= -6.08866167E+00
## EVALN #  41  F(  8.16296756E-01)= -6.08866215E+00
## EVALN #  42  F(  8.16454053E-01)= -6.08866215E+00
## EVALN #  43  F(  8.16768646E-01)= -6.08866215E+00
## EVALN #  44  F(  8.16516995E-01)= -6.08866215E+00
## EVALN #  45  F(  8.16579878E-01)= -6.08866215E+00
## EVALN #  46  F(  8.16642821E-01)= -6.08866215E+00
## EVALN #  47  F(  8.16705704E-01)= -6.08866215E+00
## EVALN #  48  F(  8.16391170E-01)= -6.08866215E+00
## EVALN #  49  F(  8.16416323E-01)= -6.08866215E+00
## EVALN #  50  F(  8.16441476E-01)= -6.08866215E+00
## EVALN #  51  F(  8.16466689E-01)= -6.08866215E+00
## EVALN #  52  F(  8.16491842E-01)= -6.08866215E+00
## EVALN #  53  F(  8.16366017E-01)= -6.08866215E+00
## EVALN #  54  F(  8.16376090E-01)= -6.08866215E+00
## EVALN #  55  F(  8.16386163E-01)= -6.08866215E+00
## EVALN #  56  F(  8.16396177E-01)= -6.08866215E+00
## EVALN #  57  F(  8.16406250E-01)= -6.08866215E+00
## EVALN #  58  F(  8.16355944E-01)= -6.08866215E+00
```

```
##  EVALN #  59  F(  8.16359997E-01)= -6.08866215E+00
##  EVALN #  60  F(  8.16363990E-01)= -6.08866215E+00
##  EVALN #  61  F(  8.16368043E-01)= -6.08866215E+00
##  EVALN #  62  F(  8.16372037E-01)= -6.08866215E+00
## OFINAL INTERVAL=(  8.16355944E-01,  8.16376090E-01)  LOWEST VALUE= -6.08866215E+00 COUNT=  62
## OTEST- COUNT=    5 NBIS=   10 U=        0.00000 V=        3.00000 TOL=   0.0000000000
##  EVALN #   1  F(  0.00000000E+00)= -5.00000000E+00
##  EVALN #   2  F(  3.00000000E+00)=  1.60000000E+01
##  EVALN #   3  F(  3.00000012E-01)= -5.57299995E+00
##  EVALN #   4  F(  6.00000024E-01)= -5.98400021E+00
## OFINAL INTERVAL=(  0.00000000E+00,  3.00000000E+00)  LOWEST VALUE= -5.98400021E+00 COUNT=   5
## OTEST- COUNT=    0 NBIS=    0 U=        0.00000 V=        0.00000 TOL=   0.0000000000
```

Pascal

Listing

```
procedure gridsrch( var lbound, ubound : real;
                    nint : integer;
                    var fmin: real;
                    var minarg: integer;
                    var changarg: integer  );

var
  j : integer;
  h, p, t : real;
  notcomp : boolean;

begin
  writeln('alg16.pas -- one-dimensional grid search');
  writeln('In gridsrch lbound=',lbound,'  ubound=',ubound);
  notcomp:=false;
  t:=fn1d(lbound, notcomp);
  writeln('  lb  f(',lbound,')=',t);
  if notcomp then halt;
  fmin:=t;
  minarg:=0;
  changarg:=0;
  h:=(ubound-lbound)/nint;
  for j:=1 to nint do

  begin
    p:=fn1d(lbound+j*h, notcomp);
    write('      f(',lbound+j*h,')=',p);
    if notcomp then halt;
    if p<fmin then
    begin
      fmin:=p; minarg:=j;
    end;
    if p*t<=0 then
    begin
      writeln(' *** sign change ***');
      changarg:=j;
    end
```

```
    else
    begin
      writeln;
    end;
    t:=p;
  end;
  writeln('Minimum so far is f(',lbound+minarg*h,')=',fmin);
  if changarg>0 then
  begin
    writeln('Sign change observed last in interval ');
    writeln(' [',lbound+(changarg-1)*h,',',lbound+changarg*h,']');
  end
  else
  begin
    writeln('Apparently no sign change in [',lbound,',',ubound,']');
  end;
end;
```

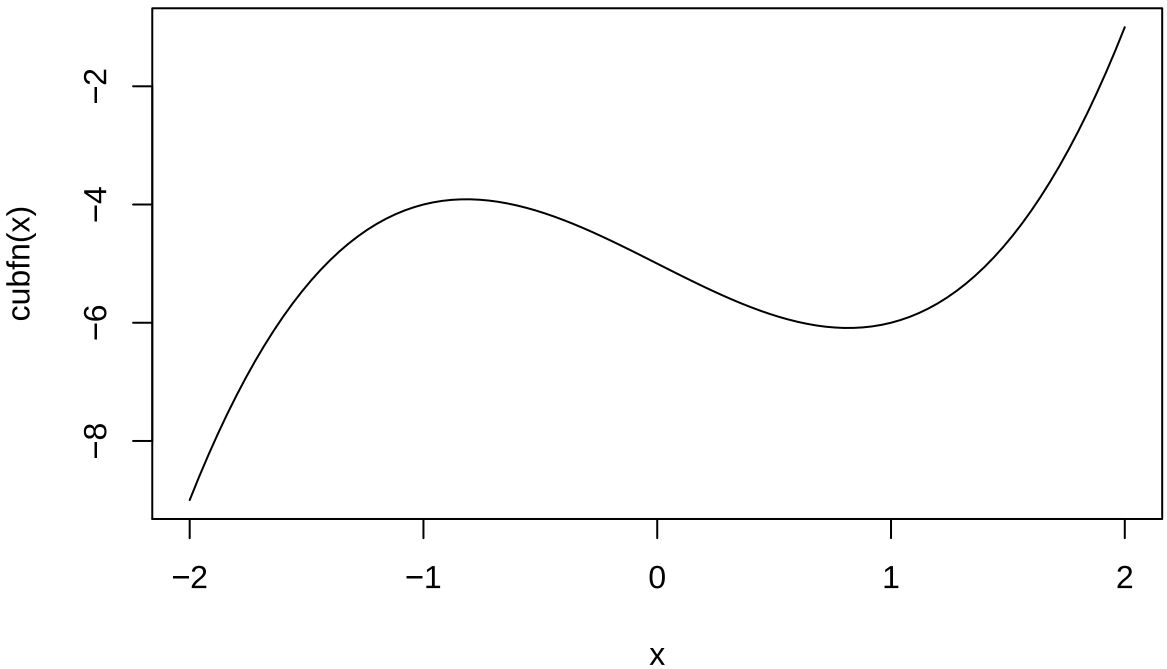
Example output

The driver for presenting the example of the Pascal version of Algorithm 16 is combined with that of Algorithm 17 below.

Algorithm 17 – Minimize a function of one parameter

It is helpful to be able to visualize a one-parameter function before trying to find a minimum. R provides a nice way to do this, and also provides (via the `Brent` method of `optim()`) a way to seek a local minimum, though we need to provide lower and upper bounds. The original Algorithm 17 from Nashlib uses a starting guess and a starting stepsize, which leads to a different approach to finding a minimum. However, the upper and lower bound approach was used in the 1990 Second Edition and its Turbo Pascal variant of the code.

```
cubfn <- function(x) { x*(x*x-2)-5}
curve(cubfn, from=-2, to=2)
```




```
res <- optim(par=0.0, fn=cubfn, method="Brent", lower=c(0), upper=c(1))
cat("Minimum proposed is f(",res$par,")=",res$value,"\n")
```

```
## Minimum proposed is f( 0.8164966 )= -6.088662
```

Fortran

Listing

```
      SUBROUTINE A17LS(B,ST,FUNS,IFN,NOCOM,IPR)
C  ALGORITHM 17 SUCCESS-FAILURE LINEAR SEARCH WITH PARABOLIC
C    INVERSE INTERPOLATION
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  B=INITIAL GUESS TO MINIMUM OF FUNCTION FUNS ALONG THE REAL LINE
C  ON OUTPUT B IS COMPUTED MINIMUM
C  ST=INITIAL STEP SIZE
C  ON OUTPUT ST CONTAINS COMPUTED MINIMUM FUNCTION VALUE
C  FUNS=NAME OF FUNCTION SUBPROGRAM
C    CALLING SEQUENCE IS      FVAL=FUNS(B,NOCOM)
C    NOCOM SET .TRUE. IF B NOT A VALID (OR DESIRABLE) ARGUMENT
C  NOCOM=LOGICAL FLAG SET .TRUE. IF INITIAL ARGUMENT INVALID
C    NORMAL RETURN FROM A17LS LEAVES NOCOM .FALSE.
C  IFN=LIMIT ON NO. OF FUNCTION EVALUATIONS (ON INPUT)
C     =NO. OF FUNCTION EVALUATIONS ACTUALLY USED (ON OUTPUT)
C  IPR=PRINTER CHANNEL IPR.GT.0 CAUSES INTERMEDIATE OUTPUT
C  STEP 0
      LOGICAL NOCOM
      INTEGER IFN,LIFN,IPR
      REAL FUNS,B,ST,A1,A2,P,S1,S0,X0,X2,BMIN,BIG,X1
C  FOR ALTERNATE TEST AT STEP 4
C     REAL EPS
C     EPS=16.0**(-5)
C  IBM VALUES
```

```
C&&&       BIG=R1MACH(2)
      BIG = 1.0E+35
      NOCOM=.FALSE.
      LIFN=IFN
      IFN=0
C  STEP CHANGE FACTORS
      A1=1.5
      A2=-0.25
C  CHECK STEPSIZE
      IF(ST.EQ.0.0)NOCOM=.TRUE.
      IF(NOCOM)RETURN
C  STEP 1
      IFN=IFN+1
      IF(IFN.GT.LIFN)GOTO 210
      P=-BIG
      P=FUNS(B,NOCOM)
      IF(IPR.GT.0)WRITE(IPR,965)IFN,B,P
 965  FORMAT(13H EVALUATION #,I4, 5H   F(,1PE16.8,2H)=,E16.8)
      IF(NOCOM)RETURN
C  STEP 2
  20  S1=P
      S0=-BIG
      X1=0.0
      BMIN=B
C  STEP 3
  30  X2=X1+ST
      B=BMIN+X2
C  STEP 4
      IF(B.EQ.BMIN+X1)GOTO 220
C  ALTERNATIVE STEP 4
C     IF(ABS(B)+EPS.EQ.ABS(BMIN)+ABS(X1)+EPS)GOTO 210
C  STEP 5
      IFN=IFN+1
      IF(IFN.GT.LIFN)GOTO 210
      NOCOM=.FALSE.
      P=FUNS(B,NOCOM)
      IF(NOCOM)GOTO 90
      IF(IPR.GT.0)WRITE(IPR,965)IFN,B,P
C  STEP 6
      IF(P.LT.S1)GOTO 100
C  STEP 7
      IF(S0.GE.S1)GOTO 110
C  STEP 8
      S0=P
      X0=X2
C  STEP 9
  90  ST=A2*ST
      GOTO 30
C  STEP 10
 100  X0=X1
      S0=S1
      X1=X2
      S1=P
```

```
      ST=A1*ST
      GOTO 30
C   STEP 11
 110  X0=X0-X1
      S0=(S0-S1)*ST
      P=(P-S1)*X0
C   STEP 12
      IF(P.EQ.S0)GOTO 180
C   STEP 13
      ST=0.5*(P*X0-S0*ST)/(P-S0)
C   STEP 14
      X2=X1+ST
      B=BMIN+X2
C   STEP 15
      IF(B.EQ.BMIN+X1)GOTO 180
C   FIXED TO JUMP TO STEP 18, NOT STEP 20 (APRIL 1989)
C   STEP 16
      IFN=IFN+1
      IF(IFN.GT.LIFN)GOTO 210
      NOCOM=.FALSE.
      P=FUNS(B,NOCOM)
      IF(NOCOM)GOTO 180
      IF(IPR.GT.0)WRITE(IPR,965)IFN,B,P
C   STEP 17
      IF(P.LT.S1)GOTO 190
C   STEP 18
 180  B=BMIN+X1
      P=S1
      GOTO 200
C   STEP 19
 190  X1=X2
C   STEP 20
 200  ST=A2*ST
      GOTO 20
 210  IFN=LIFN
 220  B=BMIN
      ST=S1
      RETURN
      END
```

Example output

```
gfortran ../fortran/dr17.f
mv ./a.out ../fortran/dr17f.run
../fortran/dr17f.run < ../fortran/dr17f.in
```

```
##  TEST A17LS STARTING POSN=        0.00000  INITIAL STEP=        1.00000
##  CONVERGED IN  19 EVALS TO F(  8.16470623E-01)= -6.08866215E+00
##
##  TEST A17LS STARTING POSN=        0.00000  INITIAL STEP=        0.10000
##  CONVERGED IN  18 EVALS TO F(  8.16489398E-01)= -6.08866215E+00
##
##  TEST A17LS STARTING POSN=        0.00000  INITIAL STEP=        0.01000
##  CONVERGED IN  27 EVALS TO F(  8.16469610E-01)= -6.08866215E+00
```

```
##
##   TEST A17LS STARTING POSN=        1.00000   INITIAL STEP=        1.00000
##   CONVERGED IN  18 EVALS TO F(  8.16750884E-01)= -6.08866215E+00
##
##   TEST A17LS STARTING POSN=        1.00000   INITIAL STEP=       -0.10000
##   CONVERGED IN  17 EVALS TO F(  8.16494286E-01)= -6.08866215E+00
##
##   TEST A17LS STARTING POSN=        1.00000   INITIAL STEP=       20.00000
##   FAILURE??
##   CONVERGED IN 100 EVALS TO F( -4.99425268E+00)= -1.19580940E+02
##
##   TEST A17LS STARTING POSN=        0.00000   INITIAL STEP=        0.00000
```

Pascal

Listing

```
procedure min1d(var bb : real;
                var st: real;
                var ifn : integer;
                var fnminval : real );



var
  a1, a2, fii, s0, s1, s2, tt0, tt1, tt2, x0, x1, x2, xii : real;
  notcomp, tripleok: boolean;

begin
  writeln('alg17.pas -- One dimensional function minimisation');
  ifn := 0;
  a1 := 1.5;
  a2 := -0.25;
  x1 := bb;
  notcomp := false;
  s0 := fn1d(x1,notcomp); ifn := ifn+1;
  if notcomp then
  begin
    writeln('*** FAILURE *** Function cannot be computed at initial point');
    halt;
  end;
  repeat
    x0 := x1;
    bb := x0;
    x1 := x0+st;
    s1 := fn1d(x1,notcomp); if notcomp then s1 := big; ifn := ifn+1;
    tripleok := false;
    if s1<s0 then
    begin
      repeat
        st := st*a1;
        x2 := x1+st;
        s2 := fn1d(x2,notcomp); if notcomp then s2 := big; ifn := ifn+1;
        if s2<s1 then
        begin
```

```
        s0 := s1; s1 := s2;
        x0 := x1; x1 := x2;
        write('Success1 ');
      end
      else
      begin
        tripleok := true;
        write('Failure1');
      end;
    until tripleok;
  end
  else
  begin
    st := a2*st;
    tt2 := s0; s0 := s1; s1 := tt2;
    tt2 := x0; x0 := x1; x1 := tt2;
    repeat
      x2 := x1+st;
      s2 := fn1d(x2,notcomp); if notcomp then s2 := big; ifn := ifn+1;
      if s2<s1 then
      begin
        s0 := s1; s1 := s2; x0 := x1; x1 := x2;
        st := st*a1;
        write('Success2 ');
      end
      else
      begin
        tripleok := true; write('Failure2');
      end;
    until tripleok;
  end;

  writeln; writeln('Triple (',x0,',',s0,')');
  writeln('       (',x1,',',s1,')');  writeln('       (',x2,',',s2,')');
  tt0 := x0-x1;
  tt1 := (s0-s1)*st; tt2 := (s2-s1)*tt0;
  if tt1<>tt2 then
  begin
    st := 0.5*(tt2*tt0-tt1*st)/(tt2-tt1);
    xii := x1+st;
    writeln('Paramin step and argument :',st,' ',xii);
    if (reltest+xii)<>(reltest+x1) then
    begin
      fii := fn1d(xii,notcomp); ifn := ifn+1;
      if notcomp then fii := big;
      if fii<s1 then
      begin
        s1 := fii; x1 := xii;
        writeln('New min f(',x1,')=',s1);
      end;
    end;
  end;
  writeln(ifn,' evalns    f(',x1,')=',s1);
```

```
    s0 := s1;
  until (bb=x1);
  writeln('Apparent minimum is f(',bb,')=',s1);
  writeln('     after ',ifn,' function evaluations');
  fnminval := s1;
end;
```

Example output

First we compile the codes.

```
fpc ../Pascal2021/dr1617.pas
# copy to run file
mv ../Pascal2021/dr1617 ../Pascal2021/dr1617p.run
```

```
## 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/dr1617.pas
## Linking ../Pascal2021/dr1617
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 282 lines compiled, 0.2 sec
```

Then we run the grid search (Algorithm 16) followed by the line search routine (Algorithm 18).

```
../Pascal2021/dr1617p.run <../Pascal2021/dr1617p.in >../Pascal2021/dr1617p.out
```

```
Enter lower bound for search  0.0000000000000000E+000
Enter upper bound for search  1.0000000000000000E+000
Enter a tolerance for search interval width  1.0000000000000000E-010
Enter the number of intervals per search (0 for no grid search) 10
alg16.pas -- one-dimensional grid search
In gridsrch lbound= 0.0000000000000000E+000  ubound= 1.0000000000000000E+000
  lb  f( 0.0000000000000000E+000)=-5.0000000000000000E+000
      f( 1.0000000000000001E-001)=-5.1989999999999998E+000
      f( 2.0000000000000001E-001)=-5.3920000000000003E+000
      f( 3.0000000000000004E-001)=-5.5730000000000004E+000
      f( 4.0000000000000002E-001)=-5.7359999999999998E+000
      f( 5.0000000000000000E-001)=-5.8750000000000000E+000
      f( 6.0000000000000009E-001)=-5.9840000000000000E+000
      f( 7.0000000000000007E-001)=-6.0570000000000004E+000
      f( 8.0000000000000004E-001)=-6.0880000000000001E+000
      f( 9.0000000000000002E-001)=-6.0709999999999997E+000
      f( 1.0000000000000000E+000)=-6.0000000000000000E+000
Minimum so far is f( 8.0000000000000004E-001)=-6.0880000000000001E+000
Apparently no sign change in [ 0.0000000000000000E+000, 1.0000000000000000E+000]
New lowest function value =-6.0880000000000001E+000 in [ 6.9999999999999996E-001, 9.0000000000000002E-00
Now call the minimiser
alg17.pas -- One dimensional function minimisation
Failure1
Triple ( 8.0000000000000004E-001,-6.0880000000000001E+000)
       ( 8.2000000000000006E-001,-6.0886320000000005E+000)
       ( 8.5000000000000009E-001,-6.0858749999999997E+000)
Paramin step and argument :-3.6032388663950385E-003  8.1639676113360504E-001
New min f( 8.1639676113360504E-001)=-6.0886620834979350E+000
4 evalns    f( 8.1639676113360504E-001)=-6.0886620834979350E+000
```

```
Failure2
Triple ( 8.1279352226721002E-001,-6.0886285697028972E+000)
       ( 8.1639676113360504E-001,-6.0886620834979350E+000)
       ( 8.1729757085020383E-001,-6.0886605358342081E+000)
Paramin step and argument : 9.9272800009582988E-005  8.1649603393361458E-001
New min f( 8.1649603393361458E-001)=-6.0886621079029020E+000
7 evalns    f( 8.1649603393361458E-001)=-6.0886621079029020E+000
Failure2
Triple ( 8.1659530673362413E-001,-6.0886620840280230E+000)
       ( 8.1649603393361458E-001,-6.0886621079029020E+000)
       ( 8.1647121573361214E-001,-6.0886621063276660E+000)
Paramin step and argument : 5.4647738345575687E-007  8.1649658041099804E-001
New min f( 8.1649658041099804E-001)=-6.0886621079036347E+000
10 evalns    f( 8.1649658041099804E-001)=-6.0886621079036347E+000
Failure2
Triple ( 8.1649712688838150E-001,-6.0886621079029046E+000)
       ( 8.1649658041099804E-001,-6.0886621079036347E+000)
       ( 8.1649644379165220E-001,-6.0886621079035885E+000)
Paramin step and argument : 6.6320070817813863E-010  8.1649658107419876E-001
13 evalns    f( 8.1649658041099804E-001)=-6.0886621079036347E+000
Apparent minimum is f( 8.1649658041099804E-001)=-6.0886621079036347E+000
     after 13 function evaluations
```

But we can run just the minimizer. Note that above we use only 13 function evaluations in the minimizer, but now use 17 (for the input used in this example). However, the grid search used 11 function evaluations prior to the call to the minimizer for a total of 24.

```
../Pascal2021/dr1617p.run <../Pascal2021/dr17p.in >../Pascal2021/dr17p.out
```

```
Enter lower bound for search  0.0000000000000000E+000
Enter upper bound for search  1.0000000000000000E+000
Enter a tolerance for search interval width  1.0000000000000000E-010
Enter the number of intervals per search (0 for no grid search) 0
Now call the minimiser
alg17.pas -- One dimensional function minimisation
Success1 Failure1
Triple ( 5.9999999999999998E-001,-5.9840000000000000E+000)
       ( 7.5000000000000000E-001,-6.0781250000000000E+000)
       ( 9.7500000000000009E-001,-6.0231406249999999E+000)
Paramin step and argument : 5.9946236559139748E-002  8.0994623655913978E-001
New min f( 8.0994623655913978E-001)=-6.0885572886751467E+000
5 evalns    f( 8.0994623655913978E-001)=-6.0885572886751467E+000
Failure2
Triple ( 8.6989247311827955E-001,-6.0815260773512261E+000)
       ( 8.0994623655913978E-001,-6.0885572886751467E+000)
       ( 7.9495967741935480E-001,-6.0875359305980759E+000)
Paramin step and argument : 6.2758433158830399E-003  8.1622207987502282E-001
New min f( 8.1622207987502282E-001)=-6.0886619233532384E+000
8 evalns    f( 8.1622207987502282E-001)=-6.0886619233532384E+000
Failure2
Triple ( 8.2249792319090587E-001,-6.0885736706691658E+000)
       ( 8.1622207987502282E-001,-6.0886619233532384E+000)
       ( 8.1465311904605209E-001,-6.0886537899407127E+000)
Paramin step and argument : 2.7201374487455200E-004  8.1649409361989733E-001
New min f( 8.1649409361989733E-001)=-6.0886621078884806E+000
```

```
11 evalns    f( 8.1649409361989733E-001)=-6.0886621078884806E+000
Failure2
Triple ( 8.1676610736477184E-001,-6.0886619299420968E+000)
       ( 8.1649409361989733E-001,-6.0886621078884806E+000)
       ( 8.1642609018367873E-001,-6.0886620957326052E+000)
Paramin step and argument : 2.4833291744350515E-006  8.1649657694907174E-001
New min f( 8.1649657694907174E-001)=-6.0886621079036347E+000
14 evalns    f( 8.1649657694907174E-001)=-6.0886621079036347E+000
Failure2
Triple ( 8.1649906027824615E-001,-6.0886621078885774E+000)
       ( 8.1649657694907174E-001,-6.0886621079036347E+000)
       ( 8.1649595611677817E-001,-6.0886621079026781E+000)
Paramin step and argument : 4.0734727830310040E-009  8.1649658102254452E-001
17 evalns    f( 8.1649657694907174E-001)=-6.0886621079036347E+000
Apparent minimum is f( 8.1649657694907174E-001)=-6.0886621079036347E+000
     after 17 function evaluations
```

Algorithm 18 – Roots of a function of one parameter

We use the same cubic polynomial for our rootfinding test as for the 1D minimizer (Algorithm 17). R has a built-in 1D rootfinder, `uniroot`. This uses ideas in Brent (1973). As of UseR!2011 in Warwick, the R multiple-precision package `Rmpfr` (Maechler (2020)) did not have a rootfinder because it needed to have a pure-R code to extend the precision. During a quite period of the conference, the author (JN) translated the C code of `uniroot` to plain R, and it is now the `unirootR` function of `Rmpfr`. The code is in the `rootoned` package at http://download.r-forge.r-project.org/src/contrib/rootoned_2018-8.28.tar.gz.

Note that this is a different algorithm to that in Nashlib. Moreover, even the Nashlib codes are not necessarily fully equivalent, as over time minor variations have crept in. We are also fairly certain that the ideas of Algorithm 18 are NOT the best for performance. They were written initially for the Data General NOVA which had very poor quality floating point (24 bit mantissa, likely no guard digit, no double precision), and with very limited storage. Thus the programming goal was reliability rather than efficiency.

```
 cubfn <- function(x) { x*(x*x-2)-5}
## curve(cubfn, from=-2, to=2)
cat("The first attempt fails -- see the plot of the function above.\n")
```

```
## The first attempt fails -- see the plot of the function above.
```

```
 res <- try(uniroot(f=cubfn, lower=0, upper=1))
```

```
## Error in uniroot(f = cubfn, lower = 0, upper = 1) :
##   f() values at end points not of opposite sign
```

```
 res <- try(uniroot(f=cubfn, lower=-3, upper=3))
 cat("Root proposed is f(",res$root,")=",res$f.root,"\n")
```

```
## Root proposed is f( 2.094555 )= 3.690185e-05
```

```
 cat("Tighter tolerance?\n")
```

```
## Tighter tolerance?
```

```
 res <- try(uniroot(f=cubfn, lower=-3, upper=3, tol=1e-10))
 cat("Root proposed is f(",res$root,")=",res$f.root,"\n")
```

```
## Root proposed is f( 2.094551 )= -7.01661e-14
```

Fortran

Listing

```
      SUBROUTINE A17LS(B,ST,FUNS,IFN,NOCOM,IPR)
C  ALGORITHM 17 SUCCESS-FAILURE LINEAR SEARCH WITH PARABOLIC
C    INVERSE INTERPOLATION
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  B=INITIAL GUESS TO MINIMUM OF FUNCTION FUNS ALONG THE REAL LINE
C  ON OUTPUT B IS COMPUTED MINIMUM
C  ST=INITIAL STEP SIZE
C  ON OUTPUT ST CONTAINS COMPUTED MINIMUM FUNCTION VALUE
C  FUNS=NAME OF FUNCTION SUBPROGRAM
C    CALLING SEQUENCE IS      FVAL=FUNS(B,NOCOM)
C    NOCOM SET .TRUE. IF B NOT A VALID (OR DESIRABLE) ARGUMENT
C  NOCOM=LOGICAL FLAG SET .TRUE. IF INITIAL ARGUMENT INVALID
C    NORMAL RETURN FROM A17LS LEAVES NOCOM .FALSE.
C  IFN=LIMIT ON NO. OF FUNCTION EVALUATIONS (ON INPUT)
C     =NO. OF FUNCTION EVALUATIONS ACTUALLY USED (ON OUTPUT)
C  IPR=PRINTER CHANNEL IPR.GT.0 CAUSES INTERMEDIATE OUTPUT
C  STEP 0
      LOGICAL NOCOM
      INTEGER IFN,LIFN,IPR
      REAL FUNS,B,ST,A1,A2,P,S1,S0,X0,X2,BMIN,BIG,X1
C  FOR ALTERNATE TEST AT STEP 4
C     REAL EPS
C     EPS=16.0**(-5)
C  IBM VALUES
C&&&       BIG=R1MACH(2)
      BIG = 1.0E+35
      NOCOM=.FALSE.
      LIFN=IFN
      IFN=0
C  STEP CHANGE FACTORS
      A1=1.5
      A2=-0.25
C  CHECK STEPSIZE
      IF(ST.EQ.0.0)NOCOM=.TRUE.
      IF(NOCOM)RETURN
C  STEP 1
      IFN=IFN+1
      IF(IFN.GT.LIFN)GOTO 210
      P=-BIG
      P=FUNS(B,NOCOM)
      IF(IPR.GT.0)WRITE(IPR,965)IFN,B,P
 965  FORMAT(13H EVALUATION #,I4, 5H   F(,1PE16.8,2H)=,E16.8)
      IF(NOCOM)RETURN
C  STEP 2
  20  S1=P
      S0=-BIG
      X1=0.0
      BMIN=B
C  STEP 3
  30  X2=X1+ST
```

```
      B=BMIN+X2
C  STEP 4
      IF(B.EQ.BMIN+X1)GOTO 220
C  ALTERNATIVE STEP 4
C     IF(ABS(B)+EPS.EQ.ABS(BMIN)+ABS(X1)+EPS)GOTO 210
C  STEP 5
      IFN=IFN+1
      IF(IFN.GT.LIFN)GOTO 210
      NOCOM=.FALSE.
      P=FUNS(B,NOCOM)
      IF(NOCOM)GOTO 90
      IF(IPR.GT.0)WRITE(IPR,965)IFN,B,P
C  STEP 6
      IF(P.LT.S1)GOTO 100
C  STEP 7
      IF(S0.GE.S1)GOTO 110
C  STEP 8
      S0=P
      X0=X2
C  STEP 9
  90  ST=A2*ST
      GOTO 30
C  STEP 10
 100  X0=X1
      S0=S1
      X1=X2
      S1=P
      ST=A1*ST
      GOTO 30
C  STEP 11
 110  X0=X0-X1
      S0=(S0-S1)*ST
      P=(P-S1)*X0
C  STEP 12
      IF(P.EQ.S0)GOTO 180
C  STEP 13
      ST=0.5*(P*X0-S0*ST)/(P-S0)
C  STEP 14
      X2=X1+ST
      B=BMIN+X2
C  STEP 15
      IF(B.EQ.BMIN+X1)GOTO 180
C  FIXED TO JUMP TO STEP 18, NOT STEP 20 (APRIL 1989)
C  STEP 16
      IFN=IFN+1
      IF(IFN.GT.LIFN)GOTO 210
      NOCOM=.FALSE.
      P=FUNS(B,NOCOM)
      IF(NOCOM)GOTO 180
      IF(IPR.GT.0)WRITE(IPR,965)IFN,B,P
C  STEP 17
      IF(P.LT.S1)GOTO 190
C  STEP 18
```

```
 180   B=BMIN+X1
       P=S1
       GOTO 200
C   STEP 19
 190   X1=X2
C   STEP 20
 200   ST=A2*ST
       GOTO 20
 210   IFN=LIFN
 220   B=BMIN
       ST=S1
       RETURN
       END
```

Example output

```
gfortran ../fortran/dr18.f
mv ./a.out ../fortran/dr18f.run
../fortran/dr18f.run < ../fortran/dr18f.in
```

```
##   TEST- COUNT=    5 NBIS=    5 U=        0.00000 V=        3.00000 TOL=   0.0000010000
##     2 EVALNS, F(  0.00000000E+00)= -5.00000000E+00  F(  3.00000000E+00)=  1.60000000E+01
##     3 EVALNS, F(  7.14285731E-01)= -6.06413984E+00  F(  3.00000000E+00)=  1.60000000E+01
##     4 EVALNS, F(  1.34249473E+00)= -5.26542187E+00  F(  3.00000000E+00)=  1.60000000E+01
##     5 EVALNS, F(  1.75290096E+00)= -3.11973000E+00  F(  3.00000000E+00)=  1.60000000E+01
##   FAILURE
##   ROOT U=  1.75290096E+00  F(U)= -3.11973000E+00 AFTER   5 EVALNS
##   TEST- COUNT=   40 NBIS=    5 U=        0.00000 V=        3.00000 TOL=   0.0000000000
##     2 EVALNS, F(  0.00000000E+00)= -5.00000000E+00  F(  3.00000000E+00)=  1.60000000E+01
##     3 EVALNS, F(  7.14285731E-01)= -6.06413984E+00  F(  3.00000000E+00)=  1.60000000E+01
##     4 EVALNS, F(  1.34249473E+00)= -5.26542187E+00  F(  3.00000000E+00)=  1.60000000E+01
##     5 EVALNS, F(  1.75290096E+00)= -3.11973000E+00  F(  3.00000000E+00)=  1.60000000E+01
##     6 EVALNS, F(  1.95638776E+00)= -1.42479300E+00  F(  3.00000000E+00)=  1.60000000E+01
##   BISECTION AT EVALN #    7
##     7 EVALNS, F(  2.04172206E+00)= -5.72261810E-01  F(  3.00000000E+00)=  1.60000000E+01
##     8 EVALNS, F(  2.04172206E+00)= -5.72261810E-01  F(  2.52086115E+00)=  5.97769737E+00
##     9 EVALNS, F(  2.08358383E+00)= -1.21660233E-01  F(  2.52086115E+00)=  5.97769737E+00
##    10 EVALNS, F(  2.09230590E+00)= -2.50315666E-02  F(  2.52086115E+00)=  5.97769737E+00
##    11 EVALNS, F(  2.09409308E+00)= -5.11503220E-03  F(  2.52086115E+00)=  5.97769737E+00
##   BISECTION AT EVALN #   12
##    12 EVALNS, F(  2.09445810E+00)= -1.04284286E-03  F(  2.52086115E+00)=  5.97769737E+00
##    13 EVALNS, F(  2.09445810E+00)= -1.04284286E-03  F(  2.30765963E+00)=  2.67364454E+00
##    14 EVALNS, F(  2.09454131E+00)= -1.13964081E-04  F(  2.30765963E+00)=  2.67364454E+00
##    15 EVALNS, F(  2.09455061E+00)= -1.00135803E-05  F(  2.30765963E+00)=  2.67364454E+00
##    16 EVALNS, F(  2.09455132E+00)= -2.38418579E-06  F(  2.30765963E+00)=  2.67364454E+00
##   BISECTION AT EVALN #   17
##   ROOT U=  2.09455156E+00  F(U)=  1.43051147E-06 AFTER  17 EVALNS
##   TEST- COUNT=   80 NBIS=    1 U=        0.00000 V=        3.00000 TOL=   0.0000000000
##     2 EVALNS, F(  0.00000000E+00)= -5.00000000E+00  F(  3.00000000E+00)=  1.60000000E+01
##   BISECTION AT EVALN #    3
##     3 EVALNS, F(  7.14285731E-01)= -6.06413984E+00  F(  3.00000000E+00)=  1.60000000E+01
##   BISECTION AT EVALN #    4
##     4 EVALNS, F(  1.85714281E+00)= -2.30903840E+00  F(  3.00000000E+00)=  1.60000000E+01
##   BISECTION AT EVALN #    5
```

```
##     5 EVALNS, F(  1.85714281E+00)= -2.30903840E+00  F(  2.42857146E+00)=  4.46647263E+00
##  BISECTION AT EVALN #    6
##     6 EVALNS, F(  1.85714281E+00)= -2.30903840E+00  F(  2.14285707E+00)=  5.53935051E-01
##  BISECTION AT EVALN #    7
##     7 EVALNS, F(  2.00000000E+00)= -1.00000000E+00  F(  2.14285707E+00)=  5.53935051E-01
##  BISECTION AT EVALN #    8
##     8 EVALNS, F(  2.07142854E+00)= -2.54737854E-01  F(  2.14285707E+00)=  5.53935051E-01
##  BISECTION AT EVALN #    9
##     9 EVALNS, F(  2.07142854E+00)= -2.54737854E-01  F(  2.10714293E+00)=  1.41536236E-01
##  BISECTION AT EVALN #   10
##    10 EVALNS, F(  2.08928585E+00)= -5.85985184E-02  F(  2.10714293E+00)=  1.41536236E-01
##  BISECTION AT EVALN #   11
##    11 EVALNS, F(  2.08928585E+00)= -5.85985184E-02  F(  2.09821439E+00)=  4.09674644E-02
##  BISECTION AT EVALN #   12
##    12 EVALNS, F(  2.09375000E+00)= -8.94165039E-03  F(  2.09821439E+00)=  4.09674644E-02
##  BISECTION AT EVALN #   13
##    13 EVALNS, F(  2.09375000E+00)= -8.94165039E-03  F(  2.09598207E+00)=  1.59802437E-02
##  BISECTION AT EVALN #   14
##    14 EVALNS, F(  2.09375000E+00)= -8.94165039E-03  F(  2.09486604E+00)=  3.51095200E-03
##  BISECTION AT EVALN #   15
##    15 EVALNS, F(  2.09430790E+00)= -2.71844864E-03  F(  2.09486604E+00)=  3.51095200E-03
##  BISECTION AT EVALN #   16
##    16 EVALNS, F(  2.09430790E+00)= -2.71844864E-03  F(  2.09458685E+00)=  3.95298004E-04
##  BISECTION AT EVALN #   17
##    17 EVALNS, F(  2.09444737E+00)= -1.16205215E-03  F(  2.09458685E+00)=  3.95298004E-04
##  BISECTION AT EVALN #   18
##    18 EVALNS, F(  2.09451723E+00)= -3.82423401E-04  F(  2.09458685E+00)=  3.95298004E-04
##  BISECTION AT EVALN #   19
##    19 EVALNS, F(  2.09451723E+00)= -3.82423401E-04  F(  2.09455204E+00)=  6.67572021E-06
##  BISECTION AT EVALN #   20
##    20 EVALNS, F(  2.09453464E+00)= -1.87873840E-04  F(  2.09455204E+00)=  6.67572021E-06
##  BISECTION AT EVALN #   21
##    21 EVALNS, F(  2.09454346E+00)= -9.01222229E-05  F(  2.09455204E+00)=  6.67572021E-06
##  BISECTION AT EVALN #   22
##    22 EVALNS, F(  2.09454775E+00)= -4.14848328E-05  F(  2.09455204E+00)=  6.67572021E-06
##  BISECTION AT EVALN #   23
##    23 EVALNS, F(  2.09454989E+00)= -1.76429749E-05  F(  2.09455204E+00)=  6.67572021E-06
##  BISECTION AT EVALN #   24
##    24 EVALNS, F(  2.09455109E+00)= -4.76837158E-06  F(  2.09455204E+00)=  6.67572021E-06
##  BISECTION AT EVALN #   25
##    25 EVALNS, F(  2.09455109E+00)= -4.76837158E-06  F(  2.09455156E+00)=  1.43051147E-06
##  BISECTION AT EVALN #   26
##  ROOT U=  2.09455156E+00  F(U)=  1.43051147E-06 AFTER  26 EVALNS
##  TEST- COUNT=   40 NBIS=    5 U=        0.00000 V=        3.00000 TOL=   0.0010000000
##     2 EVALNS, F(  0.00000000E+00)= -5.00000000E+00  F(  3.00000000E+00)=  1.60000000E+01
##     3 EVALNS, F(  7.14285731E-01)= -6.06413984E+00  F(  3.00000000E+00)=  1.60000000E+01
##     4 EVALNS, F(  1.34249473E+00)= -5.26542187E+00  F(  3.00000000E+00)=  1.60000000E+01
##     5 EVALNS, F(  1.75290096E+00)= -3.11973000E+00  F(  3.00000000E+00)=  1.60000000E+01
##     6 EVALNS, F(  1.95638776E+00)= -1.42479300E+00  F(  3.00000000E+00)=  1.60000000E+01
##  BISECTION AT EVALN #    7
##     7 EVALNS, F(  2.04172206E+00)= -5.72261810E-01  F(  3.00000000E+00)=  1.60000000E+01
##     8 EVALNS, F(  2.04172206E+00)= -5.72261810E-01  F(  2.52086115E+00)=  5.97769737E+00
##     9 EVALNS, F(  2.08358383E+00)= -1.21660233E-01  F(  2.52086115E+00)=  5.97769737E+00
##    10 EVALNS, F(  2.09230590E+00)= -2.50315666E-02  F(  2.52086115E+00)=  5.97769737E+00
```

```
##    11 EVALNS, F(  2.09409308E+00)= -5.11503220E-03  F(  2.52086115E+00)=  5.97769737E+00
##  BISECTION AT EVALN #   12
##    12 EVALNS, F(  2.09445810E+00)= -1.04284286E-03  F(  2.52086115E+00)=  5.97769737E+00
##    13 EVALNS, F(  2.09445810E+00)= -1.04284286E-03  F(  2.30765963E+00)=  2.67364454E+00
##    14 EVALNS, F(  2.09454131E+00)= -1.13964081E-04  F(  2.30765963E+00)=  2.67364454E+00
##    15 EVALNS, F(  2.09455061E+00)= -1.00135803E-05  F(  2.30765963E+00)=  2.67364454E+00
##    16 EVALNS, F(  2.09455132E+00)= -2.38418579E-06  F(  2.30765963E+00)=  2.67364454E+00
##  ROOT U=  2.09455132E+00  F(U)= -2.38418579E-06 AFTER  17 EVALNS
##  TEST- COUNT=   40 NBIS=    5 U=        0.00000 V=        1.00000 TOL=   0.0010000000
##  FAILURE
##  ROOT U=  0.00000000E+00  F(U)=  1.00000000E+00 AFTER   2 EVALNS
##  TEST- COUNT=    0 NBIS=    0 U=        0.00000 V=        0.00000 TOL=   0.0000000000
```

BASIC

The code used here was edited from one dated August 30, 1976. Changes were needed to adapt to the changed syntax of the `PRINT` statement and to allow us to run the program inside a scripted environment, but the logic is unchanged. For example, we have artificially inserted a working set of values to start the Bisection / False Position rootfinder after the grid search. The original program was designed to present the grid search so that the user could interactively choose an interval for which the endpoints had different function values to start the rootfinder.

Listing

```
5 PRINT "ENHROO AUG 30 76"
10 PRINT "GRID SEARCH"
20 READ U
30 REM PRINT "U=";U
40 READ V
50 PRINT "U=";U;"  V=";V
70 READ N9
80 PRINT "# OF POINTS";N9
100 LET H=(V-U)/N9
110 FOR I=0 TO N9
120 LET B=U+I*H
130 GOSUB 2000
140 PRINT "F(",B,")=",P
150 NEXT I
160 REM STOP
200 PRINT "ROOTFINDER"
210 READ U
220 REM PRINT "U=";U
230 READ V
240 PRINT "U=";U;"  V=";V
250 REM PRINT
260 READ N9
270 PRINT "BISECTION EVERY";N9
280 REM PRINT
290 READ E3
300 PRINT "TOLERANCE";E3
310 REM PRINT
320 GOSUB 1000
330 PRINT "ROOT: F(",B,")=",P
335 PRINT "Done!" : rem stop
```

```
340 QUIT
1000 REM BISECTION/FALSE POSITION ROOT-FINDER
1010 LET B=U
1020 GOSUB 2000
1030 LET F1=P
1040 LET B=V
1050 GOSUB 2000
1060 LET F2=P
1070 IF F1*F2<=0 THEN GOTO 1090
1075 PRINT "FUNCTIONS HAVE SAME SIGN AT BOTH ENDS OF INTERVAL"
1080 QUIT
1090 PRINT "F(";U;")=";F1;" F(";V;")=";F2
1100 LET I9=0
1110 REM FALSE POSITION
1115 PRINT "FP ";
1120 LET B=(U*F2-V*F1)/(F2-F1)
1130 IF B>U THEN GOTO 1160
1140 LET B=U
1145 LET P=F1
1150 GOTO 1320
1160 IF B<V THEN GOTO 1190
1170 LET B=V
1175 LET P=F2
1180 GOTO 1320
1190 LET I9=I9+1
1200 GOSUB 2000
1210 PRINT "ITN";I9;" U=";U;" V=";V;" F(";B;")=";P
1220 IF P*F1>0 THEN GOTO 1260
1230 LET F2=P
1240 LET V=B
1250 GOTO 1280
1260 LET F1=P
1270 LET U=B
1280 IF (V-U)<E3 THEN GOTO 1320
1290 IF N9*INT(I9/N9)<>I9 THEN GOTO 1110
1295 PRINT "BI ";
1300 LET B=(U+V)/2 : REM BETTER IS U+(V-U)*0.5
1310 GOTO 1130
1320 PRINT "CONVERGED"
1330 RETURN
2000 REM CUBIC FUNCTION TEST
2010 LET P=B*(B*B-2.0)-5.0
2020 REM NOTE USE ARGUMENT B AND RETURNED VALUE P
2090 RETURN
2200 DATA  0, 5, 10, 2, 2.5, 5, 1e-12
2300 END
```

Example output

```
bwbasic ../BASIC/a18roo.bas
```

```
## Bywater BASIC Interpreter/Shell, version 2.20 patch level 2
## Copyright (c) 1993, Ted A. Campbell
## Copyright (c) 1995-1997, Jon B. Volkoff
```

```
##
## ENHROO AUG 30 76
## GRID SEARCH
## U= 0  V= 5
## # OF POINTS 10
## F(           0            )=             -5
## F(           0.5000000    )=             -5.8750000
## F(           1            )=             -6
## F(           1.5000000    )=             -4.6250000
## F(           2            )=             -1
## F(           2.5000000    )=             5.6250000
## F(           3            )=             16
## F(           3.5000000    )=             30.8750000
## F(           4            )=             51
## F(           4.5000000    )=             77.1250000
## F(           5            )=             110
## ROOTFINDER
## U= 2  V= 2.5000000
## BISECTION EVERY 5
## TOLERANCE 0
## F( 2)= -1 F( 2.5000000)= 5.6250000
## FP ITN 1 U= 2 V= 2.5000000 F( 2.0754717)= -0.2106773
## FP ITN 2 U= 2.0754717 V= 2.5000000 F( 2.0907978)= -0.0418075
## FP ITN 3 U= 2.0907978 V= 2.5000000 F( 2.0938168)= -0.0081969
## FP ITN 4 U= 2.0938168 V= 2.5000000 F( 2.0944078)= -0.0016033
## FP ITN 5 U= 2.0944078 V= 2.5000000 F( 2.0945234)= -0.0003135
## BI ITN 6 U= 2.0945234 V= 2.5000000 F( 2.2972617)= 2.5290715
## FP ITN 7 U= 2.0945234 V= 2.2972617 F( 2.0945485)= -0.000033
## FP ITN 8 U= 2.0945485 V= 2.2972617 F( 2.0945512)= -0.0000035
## FP ITN 9 U= 2.0945512 V= 2.2972617 F( 2.0945514)= -0.0000004
## FP ITN 10 U= 2.0945514 V= 2.2972617 F( 2.0945515)= -0
## BI ITN 11 U= 2.0945515 V= 2.2972617 F( 2.1959066)= 1.196861
## FP ITN 12 U= 2.0945515 V= 2.1959066 F( 2.0945515)= -0
## FP ITN 13 U= 2.0945515 V= 2.1959066 F( 2.0945515)= -0
## FP ITN 14 U= 2.0945515 V= 2.1959066 F( 2.0945515)= -0
## FP ITN 15 U= 2.0945515 V= 2.1959066 F( 2.0945515)= -0
## BI ITN 16 U= 2.0945515 V= 2.1959066 F( 2.145229)= 0.5819023
## FP ITN 17 U= 2.0945515 V= 2.145229 F( 2.0945515)= -0
## FP ITN 18 U= 2.0945515 V= 2.145229 F( 2.0945515)= 0
## CONVERGED
## ROOT: F(      2.0945515    )=             0
## Done!
```

Pascal

Listing

Note that in this routine, we use bisection every 5 function evaluations. That is, we fix the `nbis` variable at 5. This could easily be changed to make it an input quantity.

?? Do we want to discuss why this may be useful?

```
procedure root1d(var lbound, ubound: real;
                 var ifn: integer;
                     tol : real;
```

```
                 var noroot: boolean );

var
 nbis: integer;
 b, fb, flow, fup : real;
 notcomp: boolean;

begin
  writeln('alg18.pas -- root of a function of one variable');

  notcomp := false;
  ifn := 2;
  nbis := 5;
  fup := fn1d(ubound,notcomp);
  if notcomp then halt;
  flow := fn1d(lbound,notcomp);
  if notcomp then halt;
  writeln('f(',lbound:8:5,')=',flow,'  f(',ubound:8:5,')=',fup);
  if fup*flow>0 then noroot := true else noroot := false;
  while (not noroot) and ((ubound-lbound)>tol) do
  begin

    if (nbis * ((ifn - 2) div nbis) = (ifn - 2)) then
    begin
      write('Bisect  ');
      b := lbound + 0.5*(ubound - lbound)
    end
    else
    begin
      write('False P ');
      b := (lbound*fup-ubound*flow)/(fup-flow);
    end;

    if b<=lbound then
    begin
      b := lbound;
      ubound := lbound;
    end;
    if b>=ubound then
    begin
      b := ubound; lbound := ubound;
    end;
    ifn := ifn+1;
    fb := fn1d(b, notcomp);
    if notcomp then halt;
    write(ifn,' evalns: f(',b:16,')=',fb:10);
    writeln('  width interval= ',(ubound-lbound):10);
    if (ubound-lbound)>tol then
    begin
      if fb*flow<0.0 then
      begin
        fup := fb; ubound := b;
      end
```

```
      else
      begin
        flow := fb; lbound := b;
      end;
    end;
  end;
  writeln('Converged to f(',b,')=',fb);
  writeln('  Final interval width =',ubound-lbound);
end;
```

Example output

First we compile the codes.

```
fpc ../Pascal2021/dr1618.pas
# copy to run file
mv ../Pascal2021/dr1618 ../Pascal2021/dr1618p.run
```

```
## 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/dr1618.pas
## Linking ../Pascal2021/dr1618
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 205 lines compiled, 0.2 sec
```

Then we run the grid search (Algorithm 16) followed by the Bisection / False position routine (Algorithm 18).

```
../Pascal2021/dr1618p.run <../Pascal2021/dr1618a.in >../Pascal2021/dr1618a.out
```

```
Enter lower bound for search  0.0000000000000000E+000
Enter upper bound for search  5.0000000000000000E+000
Enter the number of intervals for grid search (0 for none) 10
Enter a tolerance for root search interval width  1.0000000000000000E-010
alg16.pas -- one-dimensional grid search
In gridsrch lbound= 0.0000000000000000E+000  ubound= 5.0000000000000000E+000
  lb  f( 0.0000000000000000E+000)=-5.0000000000000000E+000
      f( 5.0000000000000000E-001)=-5.8750000000000000E+000
      f( 1.0000000000000000E+000)=-6.0000000000000000E+000
      f( 1.5000000000000000E+000)=-4.6250000000000000E+000
      f( 2.0000000000000000E+000)=-1.0000000000000000E+000
      f( 2.5000000000000000E+000)= 5.6250000000000000E+000 *** sign change ***
      f( 3.0000000000000000E+000)= 1.6000000000000000E+001
      f( 3.5000000000000000E+000)= 3.0875000000000000E+001
      f( 4.0000000000000000E+000)= 5.1000000000000000E+001
      f( 4.5000000000000000E+000)= 7.7125000000000000E+001
      f( 5.0000000000000000E+000)= 1.1000000000000000E+002
Minimum so far is f( 1.0000000000000000E+000)=-6.0000000000000000E+000
Sign change observed last in interval
 [ 2.0000000000000000E+000, 2.5000000000000000E+000]
Now try rootfinder
alg18.pas -- root of a function of one variable
f( 2.00000)=-1.0000000000000000E+000  f( 2.50000)= 5.6250000000000000E+000
Bisect  3 evalns: f( 2.25000000E+000)= 1.89E+000  width interval=  5.00E-001
False P 4 evalns: f( 2.08648649E+000)=-8.96E-002  width interval=  2.50E-001
False P 5 evalns: f( 2.09388573E+000)=-7.43E-003  width interval=  1.64E-001
```

```
False P 6 evalns: f( 2.09449668E+000)=-6.12E-004  width interval=  1.56E-001
False P 7 evalns: f( 2.09454697E+000)=-5.03E-005  width interval=  1.56E-001
Bisect  8 evalns: f( 2.17227349E+000)= 9.06E-001  width interval=  1.55E-001
False P 9 evalns: f( 2.09455129E+000)=-2.14E-006  width interval=  7.77E-002
False P 10 evalns: f( 2.09455147E+000)=-9.06E-008  width interval=  7.77E-002
False P 11 evalns: f( 2.09455148E+000)=-3.84E-009  width interval=  7.77E-002
False P 12 evalns: f( 2.09455148E+000)=-1.63E-010  width interval=  7.77E-002
Bisect  13 evalns: f( 2.13341248E+000)= 4.43E-001  width interval=  7.77E-002
False P 14 evalns: f( 2.09455148E+000)=-3.51E-012  width interval=  3.89E-002
False P 15 evalns: f( 2.09455148E+000)=-7.55E-014  width interval=  3.89E-002
False P 16 evalns: f( 2.09455148E+000)=-1.78E-015  width interval=  3.89E-002
False P 17 evalns: f( 2.09455148E+000)=-1.78E-015  width interval=  0.00E+000
Converged to f( 2.0945514815423265E+000)=-1.7763568394002505E-015
  Final interval width = 0.0000000000000000E+000
```

Let us try WITHOUT grid search first.

```
../Pascal2021/dr1618p.run <../Pascal2021/dr1618b.in >../Pascal2021/dr1618b.out
```

But we can run just the minimizer. Note that above we use only 13 function evaluations in the minimizer, but now use 17 (for the input used in this example). However, the grid search used 11 function evaluations prior to the call to the minimizer for a total of 24.

```
Enter lower bound for search  0.0000000000000000E+000
Enter upper bound for search  5.0000000000000000E+000
Enter the number of intervals for grid search (0 for none) 0
Enter a tolerance for root search interval width  1.0000000000000000E-010
Now try rootfinder
alg18.pas -- root of a function of one variable
f( 0.00000)=-5.0000000000000000E+000  f( 5.00000)= 1.1000000000000000E+002
Bisect  3 evalns: f( 2.50000000E+000)= 5.63E+000  width interval=  5.00E+000
False P 4 evalns: f( 1.17647059E+000)=-5.72E+000  width interval=  2.50E+000
False P 5 evalns: f( 1.84404318E+000)=-2.42E+000  width interval=  1.32E+000
False P 6 evalns: f( 2.04121344E+000)=-5.78E-001  width interval=  6.56E-001
False P 7 evalns: f( 2.08393696E+000)=-1.18E-001  width interval=  4.59E-001
Bisect  8 evalns: f( 2.29196848E+000)= 2.46E+000  width interval=  4.16E-001
False P 9 evalns: f( 2.09345558E+000)=-1.22E-002  width interval=  2.08E-001
False P 10 evalns: f( 2.09443873E+000)=-1.26E-003  width interval=  1.99E-001
False P 11 evalns: f( 2.09453989E+000)=-1.29E-004  width interval=  1.98E-001
False P 12 evalns: f( 2.09455029E+000)=-1.33E-005  width interval=  1.97E-001
Bisect  13 evalns: f( 2.19325938E+000)= 1.16E+000  width interval=  1.97E-001
False P 14 evalns: f( 2.09455142E+000)=-7.11E-007  width interval=  9.87E-002
False P 15 evalns: f( 2.09455148E+000)=-3.80E-008  width interval=  9.87E-002
False P 16 evalns: f( 2.09455148E+000)=-2.03E-009  width interval=  9.87E-002
False P 17 evalns: f( 2.09455148E+000)=-1.08E-010  width interval=  9.87E-002
Bisect  18 evalns: f( 2.14390543E+000)= 5.66E-001  width interval=  9.87E-002
False P 19 evalns: f( 2.09455148E+000)=-2.95E-012  width interval=  4.94E-002
False P 20 evalns: f( 2.09455148E+000)=-7.99E-014  width interval=  4.94E-002
False P 21 evalns: f( 2.09455148E+000)=-6.22E-015  width interval=  4.94E-002
False P 22 evalns: f( 2.09455148E+000)=-1.78E-015  width interval=  4.94E-002
Bisect  23 evalns: f( 2.11922846E+000)= 2.79E-001  width interval=  4.94E-002
False P 24 evalns: f( 2.09455148E+000)= 3.55E-015  width interval=  2.47E-002
Converged to f( 2.0945514815423270E+000)= 3.5527136788005009E-015
  Final interval width = 4.4408920985006262E-016
```

Algorithm 19 and 20 – Nelder-Mead minimization and Axial search

Fortran

Algorithm 19 is the Nelder-Mead Simplex direct search minimization. It is perhaps better called the Polytope method to avoid confusion with the Dantzig Simplex method for linear programming. Algorithm 20 is an axial search to escape local minima, or possibly the possible collapse of the polytope and a false convergence. The axial search has no guarantee of success, but it does sometimes avoid false results.

Listing - Algorithm 19

```
      SUBROUTINE A19NM(N,B,X,NX,NX2,NOCOM,IFN,VL,FUN,STEP,IPR)
C  ALGORITHM 19 NELDER-MEAD SIMPLEX FUNCTION MINIMIZATION
C  THIS VERSION MODIFIED 1989-04-25 IN ACCORD WITH ERRORS DISCOVERED SINCE
C  THE 1979 VERSION WAS RELEASED
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  N= NO. OF PARAMETERS
C  B= VECTOR CONTAINING STARTING POINT & RETURNING MINIMUM
C  X= WORKING ARRAY NX BY NX2
C  NX & NX2 ARE DIMENSIONS OF X.  NX.GE.(N+1), NX2.GE.(N+2)
C  NOCOM=LOGICAL FLAG SET .TRUE. IF INITIAL POINT INFEASIBLE OR STEP=0.0
C  IFN = LIMIT TO NO. OF FUNCTION EVALUATIONS (INPUT)
C      = FUNCTION EVALUATIONS USED (OUTPUT)
C  VL  = FUNCTION VALUE AT MINIMUM
C  FUN = NAME OF FUNCTION SUBROUTINE  P= FUN(N,B,NOCOM)
C  STEP= STEPSIZE FOR CONSTRUCTING INITIAL SIMPLEX
C  IPR = PRINTER CHANNEL.  PRINTING IF IPR.GT.0
C  STEP0
      LOGICAL NOCOM
      INTEGER N,NX,NX2,N1,N2,IFN,LIFN,H,L,C,IPR
C
      REAL B(N),X(NX,NX2),STEP,P,VL,SIZE,ALPHA,BETA,GAMMA,T,VH,SS,VNEXT
      C=N+2
      N1=N+1
      ALPHA=1.0
      BETA=0.5
      GAMMA=2.0
      LIFN=IFN
C  IBM VALUE
C&&&       BIG=R1MACH(2)
      BIG=1.0E+35
C  STEP 1
      NOCOM=.FALSE.
      P=FUN(N,B,NOCOM)
      IFN=1
C  PRECAUTION FOR NULL STEP
      IF(STEP.EQ.0.0)NOCOM=.TRUE.
      IF(NOCOM)RETURN
      IF(IPR.GT.0)WRITE(IPR,960)P
 960  FORMAT('0INITIAL FUNCTION VALUE=',1PE16.8)
C  STEP 2
      X(N1,1)=P
      DO 10 I=1,N
        X(I,1)=B(I)
```

```
  10  CONTINUE
      L=1
      SIZE=0.0
C  STEP 3
      DO 40 J=2,N1
C  STEP 4
        DO 20 I=1,N
          X(I,J)=B(I)
  20    CONTINUE
        T=STEP
C  STEP 5
  25    X(J-1,J)=B(J-1)+T
C  STEP 6
        IF(X(J-1,J).NE.B(J-1))GOTO 30
        T=10.0*T
        GOTO 25
C  STEP 7
  30    SIZE=SIZE+ABS(T)
C  STEP 8
  40  CONTINUE
C  STEP 9
  45  SS=SIZE
C  STEP 10
      DO 60 J=1,N1
C  STEP 11
        IF(J.EQ.L)GOTO 60
C  STEP 12
        DO 50 I=1,N
          B(I)=X(I,J)
  50    CONTINUE
      NOCOM=.FALSE.
        P=FUN(N,B,NOCOM)
        IF(NOCOM)P=BIG
        IFN=IFN+1
        X(N1,J)=P
C  STEP 13
  60  CONTINUE
C  STEP 14
  65  L=1
      H=1
C****       NEXT=1
      VL=X(N1,1)
      VH=VL
C  STEP 15
      DO 80 J=2,N1
C  STEP 16
        T=X(N1,J)
C  STEP 17
        IF(T.GE.VL)GOTO 70
        VL=T
        L=J
C  STEP 18
  70    IF(T.LT.VH)GOTO 80
```

```
C****        NEXT=H
        H=J
        VH=T
C  STEP 19
  80  CONTINUE
C  PRINTOUT
      IF(IPR.GT.0)WRITE(IPR,950)IFN,VL,VH
 950  FORMAT( 8H #EVALS=,I4,19H SIMPLEX LOW + HIGH,1P2E16.8)
      IF(IFN.GT.LIFN)GOTO 400
C  STEP 20
      IF(VL.EQ.VH)RETURN
C**** FOLLOWING STATEMENT IS NEW
      VNEXT=VL+BETA*(VH-VL)
C     THIS SETS THE FUNCTION VALUE AT THE 'NEXT TO HIGHEST POINT'
C     IN AN APPROXIMATE WAY WITHOUT THE NECESSITY OF SEARCHING
C     FOR THIS POINT ITSELF.
C  STEP 21
      DO 100 I=1,N
        T=-X(I,H)
        DO 90 J=1,N1
          T=T+X(I,J)
  90    CONTINUE
        X(I,C)=T/N
 100  CONTINUE
C  STEP 22
      DO 110 I=1,N
        B(I)=(1.0+ALPHA)*X(I,C)-ALPHA*X(I,H)
 110  CONTINUE
      NOCOM=.FALSE.
      P=FUN(N,B,NOCOM)
      IFN=IFN+1
      IF(NOCOM)P=BIG
C  STEP 23
      IF(P.LT.VL)GOTO 350
C  STEP 24
      IF(P.LT.VNEXT)GOTO 390
C  STEP 25
      IF(P.GE.VH)GOTO 270
C  STEP 26
      DO 265 I=1,N
        X(I,H)=B(I)
 265  CONTINUE
      X(N1,H)=P
C  STEP 27
 270  DO 275 I=1,N
        B(I)=(1.0-BETA)*X(I,H)+BETA*X(I,C)
 275  CONTINUE
      NOCOM=.FALSE.
      P=FUN(N,B,NOCOM)
      IFN=IFN+1
      IF(NOCOM)P=BIG
C  STEP 28
      IF(P.LT.X(N1,H))GOTO 390
```

```
C   STEP 29
      SIZE=0.0
      DO 320 J=1,N1
C   STEP 30
        IF(J.EQ.L)GOTO 320
C   STEP 31
        DO 310 I=1,N
          X(I,J)=BETA*(X(I,J)-X(I,L))+X(I,L)
          SIZE=SIZE+ABS(X(I,J)-X(I,L))
 310    CONTINUE
C   STEP 32
 320  CONTINUE
C   STEP 33
      IF(SIZE.LT.SS)GOTO 45
C   STEP 34
      GOTO 400
C   STEP 35
 350  DO 355 I=1,N
        T=GAMMA*B(I)+(1.0-GAMMA)*X(I,C)
        X(I,C)=B(I)
        B(I)=T
 355  CONTINUE
      X(N1,C)=P
C   STEP 36
      NOCOM=.FALSE.
      P=FUN(N,B,NOCOM)
      IFN=IFN+1
      IF(NOCOM)P=BIG
C   STEP 37
      IF(P.LT.X(N1,C))GOTO 390
C   STEP 38
      DO 385 I=1,N
        B(I)=X(I,C)
 385  CONTINUE
      P=X(N1,C)
C   STEP 39
 390  DO 395 I=1,N
        X(I,H)=B(I)
 395  CONTINUE
      X(N1,H)=P
      GOTO 65
C   STEP 40
 400  IF(L.EQ.0)NOCOM=.TRUE.
      IF(NOCOM)RETURN
      DO 410 I=1,N
        B(I)=X(I,L)
 410  CONTINUE
      RETURN
      END
```

Listing - Algorithm 20

```
      SUBROUTINE A20AS(N,B,FUN,P0,P,E,COUNT,IPR)
C  ALGORITHM 20  AXIAL SEARCH
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  N = ORDER OF PROBLEM = NO. OF PARAMETERS TO BE VARIED
C  B = SUPPOSED MINIMUM
C  FUN = NAME OF FUNCTION ROUTINE =   P=FUN(N,B,NOCOM)
C    WHERE NOCOM SET .TRUE. IF POINT IS INFEASIBLE
C  P0   = SUPPOSED MINIMAL FN VALUE
C  P    =  OUTPUT FUNCTION VALUE.  P.LT.P0 IF SEARCH SUCCESSFUL
C       IN SUCH A CASE  P=FUN(N,B,NOCOM)  I.E. B IS LEFT ALTERED
C  E    = NUMBER.GT.0.0 USED TO COMPUTE INCREMENT AT STEP 3
C  COUNT = NUMBER OF EVALUATIONS MADE (OUTPUT ONLY)
C  IPR =  PRINT CHANNEL.  NO PRINTING UNLESS IPR.GT.0
C  STEP 0
      LOGICAL NOCOM
      INTEGER COUNT,N,IPR,I
      REAL B(N),P0,P,T,S,E,BIG
C  IBM VALUE FOR BIG - A LARGE NUMBER USED TO GIVE A VALUE TO P
C     WHEN POINT IS NOT FEASIBLE
C&&&       BIG=R1MACH(2)
      BIG=1.0E+35
      IF(IPR.GT.0)WRITE(IPR,950)P0
 950  FORMAT(34H0AXIAL SEARCH - SUPPOSED MINIMUM =,1PE16.8,3H AT)
      IF(IPR.GT.0)WRITE(IPR,951)(I,B(I),I=1,N)
 951  FORMAT( 3H B(,I3,2H)=,1PE16.8)
      COUNT=0
C  SAFETY CHECK
      IF(E.LE.0.0)STOP
C  STEP 1
      DO 90 I=1,N
C  STEP 2
        T=B(I)
C  STEP 3
        S=E*(ABS(T)+E)
C  STEP 4
        B(I)=T+S
        NOCOM=.FALSE.
        P=FUN(N,B,NOCOM)
        COUNT=COUNT+1
        IF(NOCOM)P=BIG
        IF(IPR.GT.0)WRITE(IPR,952)I,B(I),P
 952    FORMAT( 3H B(,I3,2H)=,1PE16.8,10H FN VALUE=,E16.8)
C  STEP 5
        IF(P.LT.P0)RETURN
C  STEP 6
        B(I)=T-S
        NOCOM=.FALSE.
        P=FUN(N,B,NOCOM)
        COUNT=COUNT+1
        IF(NOCOM)P=BIG
C  STEP 7
      IF(IPR.GT.0)WRITE(IPR,952)I,B(I),P
        IF(P.LT.P0)RETURN
```

```
C  STEP 8
        B(I)=T
C  STEP 9
  90  CONTINUE
      RETURN
      END
```

Example output

The example is the Wood4 test function.

```
gfortran ../fortran/dr1920f.f
# copy to run file
mv ./a.out ../fortran/dr1920f.run
../fortran/dr1920f.run <../fortran/dr1920f.in
```

```
##  PROBLEM=WOOD4 STEPSIZE=   0.1000000015  LIMITS 1000  100
##  INITIAL POINT
##    -3.0000000000  -1.0000000000  -3.0000000000  -1.0000000000
##  CONV. TO  7.87680674E+00 IN  267
##    -0.9509993792   0.9146928787  -0.9861326814   0.9836789370
##  AXIAL SEARCH
##  AXIAL SEARCH - SUPPOSED MINIMUM =  7.87680674E+00 AT
##  B(  1)= -9.50999379E-01
##  B(  2)=  9.14692879E-01
##  B(  3)= -9.86132681E-01
##  B(  4)=  9.83678937E-01
##  B(  1)= -9.50906515E-01 FN VALUE=  7.87681150E+00
##  B(  1)= -9.51092243E-01 FN VALUE=  7.87680864E+00
##  B(  2)=  9.14782226E-01 FN VALUE=  7.87680960E+00
##  B(  2)=  9.14603531E-01 FN VALUE=  7.87680721E+00
##  B(  3)= -9.86036360E-01 FN VALUE=  7.87681150E+00
##  B(  3)= -9.86229002E-01 FN VALUE=  7.87680864E+00
##  B(  4)=  9.83775020E-01 FN VALUE=  7.87680864E+00
##  B(  4)=  9.83582854E-01 FN VALUE=  7.87680864E+00
##  FINISHED
##
##  PROBLEM=WOOD4 STEPSIZE=   1.0000000000  LIMITS 1000  100
##  INITIAL POINT
##    -3.0000000000  -1.0000000000  -3.0000000000  -1.0000000000
##  CONV. TO  1.62430347E-11 IN  772
##     0.9999979734   0.9999959469   1.0000021458   1.0000042915
##  AXIAL SEARCH
##  AXIAL SEARCH - SUPPOSED MINIMUM =  1.62430347E-11 AT
##  B(  1)=  9.99997973E-01
##  B(  2)=  9.99995947E-01
##  B(  3)=  1.00000215E+00
##  B(  4)=  1.00000429E+00
##  B(  1)=  1.00097549E+00 FN VALUE=  3.83539707E-04
##  B(  1)=  9.99020457E-01 FN VALUE=  3.82801838E-04
##  B(  2)=  1.00097346E+00 FN VALUE=  1.05207771E-04
##  B(  2)=  9.99018431E-01 FN VALUE=  1.05201711E-04
##  B(  3)=  1.00097966E+00 FN VALUE=  3.45289038E-04
##  B(  3)=  9.99024630E-01 FN VALUE=  3.44609463E-04
##  B(  4)=  1.00098181E+00 FN VALUE=  9.56556541E-05
```

```
## B( 4)=  9.99026775E-01 FN VALUE=  9.56430667E-05
## FINISHED
##
## PROBLEM=WOOD4 STEPSIZE=   0.1000000015  LIMITS  100   10
## INITIAL POINT
##   -3.0000000000  -1.0000000000  -3.0000000000  -1.0000000000
## CONV. TO  7.93660498E+00 IN  101
##   -0.9601302147   0.9080898762  -0.9928485751   0.9919158816
## AXIAL SEARCH
## AXIAL SEARCH - SUPPOSED MINIMUM =  7.93660498E+00 AT
## B( 1)= -9.60130215E-01
## B( 2)=  9.08089876E-01
## B( 3)= -9.92848575E-01
## B( 4)=  9.91915882E-01
## B( 1)= -9.60036457E-01 FN VALUE=  7.93574572E+00
## NEW STARTING POINT
##   -0.9600364566   0.9080898762  -0.9928485751   0.9919158816
## CONV. TO  7.93574572E+00 IN   11
##   -0.9600364566   0.9080898762  -0.9928485751   0.9919158816
## AXIAL SEARCH
## AXIAL SEARCH - SUPPOSED MINIMUM =  7.93574572E+00 AT
## B( 1)= -9.60036457E-01
## B( 2)=  9.08089876E-01
## B( 3)= -9.92848575E-01
## B( 4)=  9.91915882E-01
## B( 1)= -9.59942698E-01 FN VALUE=  7.93489218E+00
## NEW STARTING POINT
##   -0.9599426985   0.9080898762  -0.9928485751   0.9919158816
## FINISHED
##
## PROBLEM=WOOD4 STEPSIZE=   0.0010000000  LIMITS 1000  100
## INITIAL POINT
##   -3.0000000000  -1.0000000000  -3.0000000000  -1.0000000000
## CONV. TO  1.47082352E-13 IN  889
##    0.9999999404   0.9999998808   1.0000000000   1.0000000000
## AXIAL SEARCH
## AXIAL SEARCH - SUPPOSED MINIMUM =  1.47082352E-13 AT
## B( 1)=  9.99999940E-01
## B( 2)=  9.99999881E-01
## B( 3)=  1.00000000E+00
## B( 4)=  1.00000000E+00
## B( 1)=  1.00000095E+00 FN VALUE=  4.11746720E-10
## B( 1)=  9.99998987E-01 FN VALUE=  3.64968139E-10
## B( 2)=  1.00000083E+00 FN VALUE=  9.79859735E-11
## B( 2)=  9.99998927E-01 FN VALUE=  1.02578925E-10
## B( 3)=  1.00000095E+00 FN VALUE=  3.28474664E-10
## B( 3)=  9.99999046E-01 FN VALUE=  3.28474664E-10
## B( 4)=  1.00000095E+00 FN VALUE=  8.89365040E-11
## B( 4)=  9.99999046E-01 FN VALUE=  9.34385000E-11
## FINISHED
##
```

Algorithm 21 – Variable metric minimization

Fortran

Listing

```
      SUBROUTINE A21VM(N,B,BH,NBH,X,C,G,T,IFN,IG,NOCOM,IPR,P0,FUN,DER)
C  ALGORITHM 21 VARIABLE METRIC FUNCTION MINIMIZATION
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  N = NO. OF PARAMETERS TO BE ADJUSTED
C  B = INITIAL SET OF PARAMETERS (INPUT)
C    = MINIMUM  (OUTPUT)
C  BH= WORKING ARRAY
C  NBH= FIRST DIMENSION OF BH
C  X,C,G,T = WORKING VECTORS OF LENGTH AT LEAST N
C  ON OUTPUT G CONTAINS LAST GRADIENT EVALUATED
C  IFN= COUNT OF FUNCTION EVALUATIONS USED
C     = LIMIT ON THESE (INPUT)
C  IG = COUNT OF GRADIENT EVALUATIONS USED
C  NOCOM = LOGICAL FLAG SET .TRUE. IF INITIAL POINT INFEASIBLE
C  IPR = PRINTER CHANNEL.  PRINTING ONLY IF IPR.GT.0
C  P0 = MINIMAL FUNCTION VALUE
C  FUN = NAME OF FUNCTION SUBROUTINE
C  DER = NAME OF DERIVATIVE SUBROUTINE
C     CALLING SEQUENCE   P=FUN(N,B,NOCOM) -- OTHER INFO. PASSED
C     CALLING SEQUENCE   CALL DER(N,B,G)  --  THROUGH COMMON
C  STEP 0
      LOGICAL NOCOM
      INTEGER N,NBH,IFN,IG,IPR,ILAST,I,J,COUNT
      REAL B(N),BH(NBH,N),X(N),C(N),G(N),T(N),P0,W,TOL,K,S,D1,D2,P
      IG=0
      LIFN=IFN
      IFN=0
      W=0.2
      TOL=0.0001
C  STEP 1
      NOCOM=.FALSE.
      P0=FUN(N,B,NOCOM)
      IFN=IFN+1
      IF(NOCOM)RETURN
C  STEP 2  - ASSUME DERIVATIVES CAN BE COMPUTED IF FUNCTION CAN
      CALL DER(N,B,G)
      IG=IG+1
C  STEP 3
  30  DO 35 I=1,N
        DO 32 J=1,N
          BH(I,J)=0.0
  32    CONTINUE
        BH(I,I)=1.0
  35  CONTINUE
      ILAST=IG
C  STEP 4
  40  IF(IPR.GT.0)WRITE(IPR,950)IG,IFN,P0
 950  FORMAT( 6H AFTER,I4,8H GRAD. &,I4,22H FN EVALUATIONS, FMIN=,
```

```
     *1PE16.8)
      DO 45 I=1,N
        X(I)=B(I)
        C(I)=G(I)
  45  CONTINUE
C  STEP 5
      D1=0.0
      DO 55 I=1,N
        S=0.0
        DO 53 J=1,N
          S=S-BH(I,J)*G(J)
  53    CONTINUE
        T(I)=S
        D1=D1-S*G(I)
  55  CONTINUE
C  STEP 6
      IF(D1.GT.0.0)GOTO 70
      IF(ILAST.EQ.IG)GOTO 180
      GOTO 30
  70  K=1.0
C  STEP 7
C  STEP 8
  80  COUNT=0
      DO 85 I=1,N
        B(I)=X(I)+K*T(I)
        IF(B(I).EQ.X(I))COUNT=COUNT+1
  85  CONTINUE
C  STEP 9
      IF(COUNT.LT.N)GOTO 100
      IF(ILAST.EQ.IG)GOTO 180
      GOTO 30
C  STEP 10
 100  IFN=IFN+1
      IF(IFN.GT.LIFN)GOTO 175
      P=FUN(N,B,NOCOM)
      IF(.NOT.NOCOM)GOTO 110
      K=W*K
      GOTO 80
C  STEP 11
 110  IF(P.LT.P0-D1*K*TOL)GOTO 120
      K=W*K
      GOTO 80
 120  P0=P
      IG=IG+1
      CALL DER(N,B,G)
C  STEP 13
      D1=0.0
      DO 135 I=1,N
        T(I)=K*T(I)
        C(I)=G(I)-C(I)
        D1=D1+T(I)*C(I)
 135  CONTINUE
C  STEP 14
```

```
      IF(D1.LE.0.0)GOTO 30
C  STEP 15
      D2=0.0
      DO 156 I=1,N
        S=0.0
        DO 154 J=1,N
          S=S+BH(I,J)*C(J)
 154    CONTINUE
        X(I)=S
        D2=D2+S*C(I)
 156  CONTINUE
C  STEP 16
      D2=1.0+D2/D1
      DO 165 I=1,N
        DO 164 J=1,N
          BH(I,J)=BH(I,J)-(T(I)*X(J)+X(I)*T(J)-D2*T(I)*T(J))/D1
 164    CONTINUE
 165  CONTINUE
C  STEP 17
      GOTO 40
C  RESET B IN CASE FN EVALN LIMIT REACHED
C  OUT OF EVALUATIONS!  (mod 2021-2-12)
 175  IFN=-IFN
C  SET COUNT OF FUNCTIONS NEGATIVE IF LIMIT REACHED
      DO 177 I=1,N
        B(I)=X(I)
 177  CONTINUE
 180  IF(IPR.LE.0)RETURN
      WRITE(IPR,951)
 951  FORMAT(10H CONVERGED)
      WRITE(IPR,950)IG,IFN,P0
      RETURN
      END
```

Example output

We use the WOOD4 function from Nash and Walker-Smith (1987), page 421 from different starting points. The code is set up to return with a negative count of function evaluations if the pre-set limit is reached. This is tested in the first example case.

```
gfortran ../fortran/dr21f.f
mv ./a.out ../fortran/dr21f.run
../fortran/dr21f.run < ../fortran/dr21f.in
```

```
##  PROBLEM=WOOD4 STEPSIZE=   0.1000000015  LIMITS    5    5
##  INITIAL POINT
##    -3.0000000000  -1.0000000000  -3.0000000000  -1.0000000000
##  CONV. TO  1.91920000E+04 IN   -6 Fns and    1 Grads
##    -3.0000000000  -1.0000000000  -3.0000000000  -1.0000000000
##
##  PROBLEM=WOOD4 STEPSIZE=   0.1000000015  LIMITS 1000  100
##  INITIAL POINT
##     0.8999999762   0.8999999762   0.8999999762   0.8999999762
##  CONV. TO  0.00000000E+00 IN   21 Fns and   11 Grads
##     1.0000000000   1.0000000000   1.0000000000   1.0000000000
```

```
##
##  PROBLEM=WOOD4 STEPSIZE=   0.1000000015  LIMITS 1000  100
##  INITIAL POINT
##    -3.0000000000  -1.0000000000  -3.0000000000  -1.0000000000
##  CONV. TO  0.00000000E+00 IN   64 Fns and  45 Grads
##     1.0000000000   1.0000000000   1.0000000000   1.0000000000
##
##  PROBLEM=WOOD4 STEPSIZE=   0.1000000015  LIMITS  100   10
##  INITIAL POINT
##    -3.0000000000  -1.0000000000  -3.0000000000  -1.0000000000
##  CONV. TO  0.00000000E+00 IN   64 Fns and  45 Grads
##     1.0000000000   1.0000000000   1.0000000000   1.0000000000
##
```

Pascal

Listing

```
procedure vmmin(n: integer;
            var Bvec, X: rvector;
            var Fmin: real;
                Workdata: probdata;
            var fail: boolean;
            var intol: real);

const
  Maxparm = 25;
  stepredn = 0.2;
  acctol = 0.0001;
  reltest = 10.0;

var
  accpoint  : boolean;
  B         : array[1..Maxparm, 1..Maxparm] of real;

  c         : rvector;
  count     : integer;
  D1, D2     : real;
  f         : real;
  funcount  : integer;
  g         : rvector;
  gradcount : integer;
  gradproj  : real;
  i, j       : integer;
  ilast     : integer;
  notcomp   : boolean;
  s         : real;
  steplength: real;
  t         : rvector;

begin
  writeln('alg21.pas -- version 2 1988-03-24');
  writeln('  Variable metric function minimiser');
  fail:=false;
```

```
f:=fminfn(n, Bvec, Workdata, notcomp);
if notcomp then
begin
  writeln('**** Function cannot be evaluated at initial parameters ****');
  fail := true;
end
else
begin
  Fmin:=f;
  funcount:=1;
  gradcount:=1;
  fmingr(n, Bvec, Workdata, g);
  ilast:=gradcount;

  repeat
    if ilast=gradcount then
    begin
      for i:=1 to n do
      begin
        for j:=1 to n do B[i, j]:=0.0; B[i, i]:=1.0;
      end;
    end;
    writeln(gradcount,' ', funcount,' ', Fmin);
    write('parameters ');
    for i:=1 to n do write(Bvec[i]:10:5,' ');
    writeln;
    for i:=1 to n do
    begin
      X[i]:=Bvec[i];
      c[i]:=g[i];
    end;

    gradproj:=0.0;
    for i:=1 to n do
    begin
      s:=0.0;
      for j:=1 to n do s:=s-B[i, j]*g[j];
      t[i]:=s; gradproj:=gradproj+s*g[i];
    end;

    if gradproj<0.0 then {!! note change to floating point}
    begin
      steplength:=1.0;

      accpoint:=false;
      repeat
        count:=0;
        for i:=1 to n do
        begin
          Bvec[i]:=X[i]+steplength*t[i];
          if (reltest+X[i])=(reltest+Bvec[i]) then count:=count+1;
        end;
        if count<n then
```

```
    begin
      f:=fminfn(n, Bvec, Workdata, notcomp);
      funcount:=funcount+1;
      accpoint:=(not notcomp) and (f<=Fmin+gradproj*steplength*acctol);

      if not accpoint then
      begin
        steplength:=steplength*stepredn; write('*');
      end;
    end;
  until (count=n) or accpoint;
  if count<n then
  begin
    Fmin:=f;
    fmingr(n, Bvec, Workdata, g);
    gradcount:=gradcount+1;
    D1:=0.0;
    for i:=1 to n do
    begin
      t[i]:=steplength*t[i]; c[i]:=g[i]-c[i];
      D1:=D1+t[i]*c[i];
    end;
    if D1>0 then
    begin
      D2:=0.0;
      for i:=1 to n do
      begin
        s:=0.0;
        for j:=1 to n do s:=s+B[i, j]*c[j];
        X[i]:=s; D2:=D2+s*c[i];
      end;
      D2:=1.0+D2/D1;
      for i:=1 to n do
      begin
        for j:=1 to n do
        begin
          B[i, j]:=B[i, j]-(t[i]*X[j]+X[i]*t[j]-D2*t[i]*t[j])/D1;
        end;
      end;
    end
    else
    begin
      writeln(' UPDATE NOT POSSIBLE');
      ilast:=gradcount;
    end;
  end
  else
  begin
    if ilast<gradcount then
    begin
      count:=0;
      ilast:=gradcount;
    end;
```

```
        end;
      end
      else
      begin
          writeln('UPHILL SEARCH DIRECTION');
          count:=0; {!! order of statements}
          if ilast=gradcount then count:=n else ilast:=gradcount;
          {!! Resets Hessian inverse if it has not just been set,
              otherwise forces a convergence.}
      end;
    until (count=n) and (ilast=gradcount);
  end;
```

Example output

```
fpc ../Pascal2021/dr21p.pas
# copy to run file
mv ../Pascal2021/dr21p ../Pascal2021/dr21p.run
../Pascal2021/dr21p.run >../Pascal2021/dr21p.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/dr21p.pas
## Linking ../Pascal2021/dr21p
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 300 lines compiled, 0.1 sec
```

```
Function: Rosenbrock Banana Valley
Classical starting point (-1.2,1)
alg21.pas -- version 2 1988-03-24
  Variable metric function minimiser
1 1  2.4199999999999996E+001
parameters   -1.20000    1.00000
****2 6  2.0227123078849889E+001
parameters   -0.85504    1.14080
** UPDATE NOT POSSIBLE
3 9  8.6067782779004922E+000
parameters   -0.60773    0.61474
****4 14  3.1229949299889501E+000
parameters   -0.69803    0.53621
* UPDATE NOT POSSIBLE
5 16  2.8306330601569476E+000
parameters   -0.59875    0.41090
****6 21  2.6346631298119196E+000
parameters   -0.61371    0.39413
*7 23  2.0069332556024175E+000
parameters   -0.35320    0.08283
8 24  1.8900239540129975E+000
parameters   -0.37129    0.12807
9 25  1.5197524557405362E+000
parameters   -0.20093    0.01253
10 26  1.3673830993993661E+000
parameters   -0.06129   -0.04534
```

```
11 27  1.0132787128893144E+000
parameters   -0.00588   -0.00381
12 28  8.5658102384992962E-001
parameters    0.16551   -0.01263
13 29  7.3080952386292142E-001
parameters    0.14949    0.01372
14 30  5.7229570813964525E-001
parameters    0.24999    0.05260
*15 32  5.1681338160416435E-001
parameters    0.29569    0.07302
16 33  4.5862371286805992E-001
parameters    0.47399    0.18201
17 34  3.3070213691470968E-001
parameters    0.42495    0.18096
18 35  2.6642418536041679E-001
parameters    0.48511    0.23171
*19 37  2.1612305587786573E-001
parameters    0.55338    0.29332
20 38  1.8365009757181658E-001
parameters    0.63682    0.38279
21 39  1.2824050470333978E-001
parameters    0.65485    0.41928
22 40  7.6019350519806725E-002
parameters    0.74188    0.54069
23 41  4.2301248903545835E-002
parameters    0.80151    0.63703
24 42  2.3260037206616860E-002
parameters    0.86577    0.74232
25 43  1.5408039220460280E-002
parameters    0.92719    0.86974
*26 45  5.7254131019014963E-003
parameters    0.95373    0.91559
*27 47  6.8043571662395629E-005
parameters    0.99175    0.98360
28 48  5.1523512161471316E-005
parameters    0.99616    0.99173
29 49  3.3792415462109564E-005
parameters    0.99515    0.99001
30 50  2.1490571945237054E-005
parameters    0.99563    0.99112
31 51  4.4455578169014791E-006
parameters    0.99795    0.99595
32 52  3.6389350710078493E-007
parameters    0.99957    0.99918
33 53  6.2896479106191278E-009
parameters    1.00001    1.00004
34 54  1.3220504661016668E-010
parameters    1.00001    1.00002
35 55  6.0317647945407729E-014
parameters    1.00000    1.00000
36 56  3.5872507739270852E-017
parameters    1.00000    1.00000
37 57  6.9434649532294602E-022
```

```
parameters    1.00000    1.00000
38 58  1.6997351731715904E-026
parameters    1.00000    1.00000
39 59  1.2325951644078309E-032
parameters    1.00000    1.00000
39 59  1.2325951644078309E-032
parameters    1.00000    1.00000
Exiting from alg21.pas variable metric minimiser
    59 function evaluations used
    39 gradient evaluations used

 Minimum function value found = 1.2325951644078309E-032
 At parameters
 Bvec[1]= 1.0000000000000000E+000
 Bvec[2]= 1.0000000000000000E+000
```

Algorithm 22 – Conjugate gradients minimizers

Fortran

Listing

```
      SUBROUTINE A22CGM(N,B,FUN,DER,NOCOM,IPR,IFN,IG,EPS,G,X,T,C,P0)
C  ALGORITHM 22 CONJUGATE GRADIENTS FUNCTION MINIMISATION
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C  MINIMISE FUNCTION  FUN(N,B,NOCOM) W.R.T. B(I),I=1,2,...,N
C  NOCOM IS A LOGICAL FLAG SET .TRUE. IF INITIAL POINT INFEASIBLE
C  B  IS INITIAL POINT & FINAL APPROXIMATE MINIMUM
C  FUN & DER ARE THE NAMES OF FUNCTION AND DERIVATIVE SUB-PROGRAMS
C     SEE  BELOW FOR CALLING SEQUENCES
C  IPR IS PRINT CHANNEL    IPR.GT.0  FOR PRINTING
C  IFN IS NUMBER OF FN EVALUATIONS USED (OUTPUT)
C      IS LIMIT ON EVALUATIONS (INPUT)
C  IG  IS NUMBER OF DERIVATIVE EVALUATIONS
C  EPS IS MACHINE PRECISION
C  G,X,T,C ARE WORKING VECTORS
C  STEP 0
      LOGICAL NOCOM, ACCPNT
      INTEGER N,IPR,IFN,LIM,IG,COUNT,ITN,I,IFNL
      REAL B(N),X(N),G(N),T(N),C(N),P1,P0,P,G1,G2,K,MSTEP,STEP,EPS,TOL
      REAL A1,A2,T2,GRPR,G3,LSTEP,RELTST,ACCTOL,STREDN,NUSTEP,SETSTP
      STREDN=0.2
      ACCTOL=0.0001
      RELTST=10.0
      SETSTP=1.7
      LIM=IFN
      IFN=0
      IG=0
      TOL=N*EPS*SQRT(EPS)
      LSTEP=1.0
C  STEP 1
      NOCOM=.FALSE.
      P0=FUN(N,B,NOCOM)
```

```
      IF(NOCOM)RETURN
      IFN=IFN+1
C  STEP 2
  20  DO 25 I=1,N
        C(I)=0.0
        T(I)=0.0
  25  CONTINUE
C  STEP 3
  30  DO 270 ITN=1,N
        IF(IPR.GT.0)WRITE(IPR,973)IFN,IG,P0
 973    FORMAT( 1H ,I4,11H FUNCTION &,I4,32H DERIVATIVE EVALUATIONS --
     * P0=,1PE16.8)
C  STEP 4
        CALL DER(N,B,G)
        IG=IG+1
C  STEP 5
        G1=0.0
        G2=0.0
        DO 55 I=1,N
          X(I)=B(I)
C****  POLAK RIBIERE FORMULAS
C****            G1=G1+G(I)*(G(I)-C(I))
C****            G2=G2+C(I)**2
C****  FLETCHER REEVES FORMULAS -- CURRENTLY ACTIVE
          G1=G1+G(I)*G(I)
          G2=G2+C(I)*C(I)
C****  BEALE SORENSON FORMULAS
C****          G1=G1+G(I)*(G(I)-C(I))
C****          G2=G2+T(I)*(G(I)-C(I))
          C(I)=G(I)
  55    CONTINUE
C  STEP 6
        IF(G1.GT.TOL)GOTO 70
C  CHECK NOT G2, ALSO SIGN
C
        IF(ITN.EQ.1)RETURN
C  STEP 7
        G3=1.0
  70    IF(G2.GT.0.0)G3=G1/G2
C        WRITE(IPR,8003)G1,G2,G3
C 8003   FORMAT(' G1,G2,G3 ',1P3E16.8)
C  STEP 8
        GRPR=0.0
        T2=0.0
        DO 85 I=1,N
          T(I)=T(I)*G3-G(I)
          T2=T2+T(I)**2
          GRPR=GRPR+T(I)*G(I)
  85    CONTINUE
C        WRITE(IPR,8001)T2,GRPR
C 8001   FORMAT(' AT 85 T2=',1PE16.8,'   GRPR=',E16.8)
C  STEP 9
        STEP=LSTEP
```

```
C****  STEP ALONG SEARCH DIRECTION -- STEP 10
        ACCPNT=.FALSE.
C   DON'T HAVE A GOOD POINT YET
        IFNL=IFN
C   RECORDS LAST FUNCTION COUNT
C  STEP 10
  90    COUNT=0
        DO 105 I=1,N
          B(I)=X(I)+STEP*T(I)
          IF(RELTST+B(I).EQ.RELTST+X(I))COUNT=COUNT+1
 105    CONTINUE
C  STEP 11
C        WRITE(IPR,8002)COUNT
C 8002   FORMAT(' AT 105 COUNT =',I4)
        IF(COUNT.LT.N)GOTO 120
        IF(IFN.GT.IFNL)GOTO 120
        STEP=10.0*STEP
C ??? NEED TO GET RID OF UNUSED VARIABLES
C      STEP IS TOO SMALL ON FIRST TRY
       GOTO 90
C  STEP 12
 120   P=FUN(N,B,NOCOM)
       IFN=IFN+1
       ACCPNT = (.NOT.NOCOM).AND.(P.LE.P0+GRPR*STEP*ACCTOL)
       IF (ACCPNT) GOTO 160
       STEP=STREDN*STEP
       WRITE(IPR,974)
  974  FORMAT(1H+,'*')
       GOTO 90
  160  NUSTEP=2.0*((P-P0)-GRPR*STEP)
       IF(NUSTEP.LE.0.0) GOTO 195
       NUSTEP=-GRPR*STEP*STEP/NUSTEP
       DO 170 I=1,N
          B(I)=X(I)+NUSTEP*T(I)
  170  CONTINUE
       P0=P
       NOCOM=.FALSE.
       P=FUN(N,B,NOCOM)
       IFN=IFN+1
C  ??? CHECK NOCOMP ???
       IF(P.GE.P0)GOTO 180
       P0=P
       WRITE(IPR,975)
C  975  FORMAT(1H+,' I< ')
  975  FORMAT(' INTERPOLATION SUCCEEDED')
       GOTO 195
  180  DO 190 I=1,N
         B(I)=X(I)+STEP*T(I)
C      RESETS THE PARAMETERS TO THEIR 'BEST' VALUES
  190  CONTINUE
       WRITE(IPR,976)
  976  FORMAT(' INTERPOLATION FAILED')
C  976  FORMAT(1H+,' I> ')
```

```
  195  LSTEP=SETSTP*STEP
       IF(LSTEP.GT.1.0)LSTEP=1.0
C  CAN PLACE A LIMIT ON FUNCTION EVALUATIONS HERE
C  modification 2021-2-12. Chance IFN to -IFN
       IF(IFN.LT.LIM) GOTO 270
       IFN=-IFN
       RETURN
 270  CONTINUE
C END OF INNER CYCLE
      GOTO 20
      END
```

Example output

We use the WOOD4 function from Nash and Walker-Smith (1987), page 421 from different starting points. The code is set up to return with a negative count of function evaluations if the pre-set limit is reached.

??? need to explain and maybe clean up output

```
gfortran ../fortran/dr22f.f
mv ./a.out ../fortran/dr22f.run
../fortran/dr22f.run < ../fortran/dr22f.in
```

```
## OPROBLEM=WOOD4 STEPSIZE=   0.1000000015  LIMITS 1000  100
## OINITIAL POINT
##     0.8999999762   0.8999999762   0.8999999762   0.8999999762
## OCONV. TO  2.09589290E-11 IN    49 FN EVALS   17 DERIVS.
##     1.0000022650   1.0000046492   0.9999979138   0.9999958277
##
## OPROBLEM=WOOD4 STEPSIZE=   0.1000000015  LIMITS 1000  100
## OINITIAL POINT
##    -3.0000000000  -1.0000000000  -3.0000000000  -1.0000000000
## OCONV. TO  2.08058015E-10 IN   205 FN EVALS   85 DERIVS.
##     1.0000076294   1.0000152588   0.9999924302   0.9999848604
##
## OPROBLEM=WOOD4 STEPSIZE=   0.1000000015  LIMITS  100   10
## OINITIAL POINT
##    -3.0000000000  -1.0000000000  -3.0000000000  -1.0000000000
## OCONV. TO  5.73047146E-06 IN  -100 FN EVALS   41 DERIVS.
##     1.0011516809   1.0023083687   0.9987361431   0.9974750280
##
```

BASIC

The original BASIC version of Algorithm 22 seems to have been lost to time. However, the work towards Nash and Walker-Smith (1987) created a new Conjugate Gradients minimizer based on Algorithm 22 but with the addition of code allowing both bounds and masks constraints. The code is available from archive.org at ??

Pascal

Listing

```
procedure cgmin(n: integer;
          var Bvec, X: rvector;
```

```
          var Fmin: real;
            Workdata: probdata;
          var fail: boolean;
          var intol: real);

type
  methodtype= (Fletcher_Reeves, Polak_Ribiere, Beale_Sorenson);

const
  Maxparm = 25;
  stepredn = 0.2;
  acctol = 0.0001;
  reltest = 10.0;

var
  accpoint  : boolean;
  c         : rvector;
  count     : integer;
  cycle     : integer;
  cyclimit  : integer;
  f         : real;
  funcount  : integer;
  g         : rvector;
  G1, G2     : real;
  G3, gradproj     : real;
  gradcount : integer;
  i, j       : integer;
  method    : methodtype;
  newstep   : real;
  notcomp   : boolean;
  oldstep   : real;
  s         : real;
  setstep   : real;
  steplength: real;
  t         : rvector;
  tol       : real;

begin
  writeln('alg22.pas -- Nash Algorithm 22 version 2 1988-03-24');
  writeln('  Conjugate gradients function minimiser');
  writeln('Steplength saving factor multiplies best steplength found at the');
  writeln('  end of each iteration as a starting value for next search');
  write('Enter a steplength saving factor (sugg. 1.7) -- setstep ');
  readln(setstep);
  writeln(setstep);
  write('Choose method (1=FR, 2=PR, 3=BS) ');
  readln(i); writeln(i);
  case i of
    1: method:=Fletcher_Reeves;
    2: method:=Polak_Ribiere;
    3: method:=Beale_Sorenson;
    else halt;
  end;
```

```pascal
case method of
  Fletcher_Reeves: writeln('Method: Fletcher Reeves');
  Polak_Ribiere: writeln('Method: Polak Ribiere');
  Beale_Sorenson: writeln('Method: Beale Sorenson');
end;
fail:=false;
cyclimit:=n;
if intol<0.0 then intol:=Calceps;
tol:=intol*n*sqrt(intol);

writeln('tolerance used in gradient test=', tol);
f:=fminfn(n, Bvec, Workdata, notcomp);
if notcomp then
begin
  writeln('**** Function cannot be evaluated at initial parameters ****');
  fail := true;
end
else
begin
  Fmin:=f;
  funcount:=1;
  gradcount:=0;
  repeat
    for i:=1 to n do
    begin
      t[i]:=0.0;
      c[i]:=0.0;
    end;
    cycle:=0;
    oldstep:=1.0;
    count:=0;
    repeat
      cycle:=cycle+1;
      writeln(gradcount, ' ', funcount, ' ', Fmin);
      write('parameters ');
      for i:=1 to n do
      begin
        write(Bvec[i]:10:5, ' ');
        if (7 * (i div 7) = i) and (i<n) then writeln;
      end;
      writeln;
      gradcount:=gradcount+1;
      fmingr(n, Bvec, Workdata, g);
      G1:=0.0; G2:=0.0;
      for i:=1 to n do
      begin
        X[i]:=Bvec[i];
        case method of
          Fletcher_Reeves: begin
            G1:=G1+sqr(g[i]); G2:=G2+sqr(c[i]);
          end;
          Polak_Ribiere  : begin
            G1:=G1+g[i]*(g[i]-c[i]); G2:=G2+sqr(c[i]);
```

```
      end;
      Beale_Sorenson : begin
        G1:=G1+g[i]*(g[i]-c[i]); G2:=G2+t[i]*(g[i]-c[i]);
      end;
    end;
    c[i]:=g[i];
  end;
  if G1>tol then
  begin
    if G2>0.0 then G3:=G1/G2 else G3:=1.0;
    gradproj:=0.0;
    for i:=1 to n do
    begin
      t[i]:=t[i]*G3-g[i]; gradproj:=gradproj+t[i]*g[i];
    end;
    steplength:=oldstep;

    accpoint:=false;
    repeat
      count:=0;
      for i:=1 to n do
      begin
        Bvec[i]:=X[i]+steplength*t[i];
        if (reltest+X[i])=(reltest+Bvec[i]) then count:=count+1;
      end;
      if count<n then
      begin
        f:=fminfn(n, Bvec, Workdata, notcomp);
        funcount:=funcount+1;
        accpoint:=(not notcomp) and (f<=Fmin+gradproj*steplength*acctol);

        if not accpoint then
        begin
          steplength:=steplength*stepredn;
          write('*');
        end;
      end;
    until (count=n) or accpoint;
    if count<n then
    begin
      newstep:=2*((f-Fmin)-gradproj*steplength);
      if newstep>0 then
      begin
        newstep:=-gradproj*sqr(steplength)/newstep;
        for i:=1 to n do
        begin
          Bvec[i]:=X[i]+newstep*t[i];
        end;
        Fmin:=f;
        f:=fminfn(n, Bvec, Workdata, notcomp);
        funcount:=funcount+1;
        if f<Fmin then
        begin
```

```
                Fmin:=f; write(' i< ');
              end
              else
              begin
                write(' i> ');
                for i:=1 to n do Bvec[i]:=X[i]+steplength*t[i];
              end;
            end;
          end;
        end;
        oldstep:=setstep*steplength;
        if oldstep>1.0 then oldstep:=1.0;
      until (count=n) or (G1<=tol) or (cycle=cyclimit);

    until (cycle=1) and ((count=n) or (G1<=tol));

  end;
  writeln('Exiting from Alg22.pas conjugate gradients minimiser');
  writeln('    ', funcount, ' function evaluations used');
  writeln('    ', gradcount, ' gradient evaluations used');
end;
```

Example output

```
fpc ../Pascal2021/dr22p.pas
# copy to run file
mv ../Pascal2021/dr22p ../Pascal2021/dr22p.run
../Pascal2021/dr22p.run <../Pascal2021/dr22p.in >../Pascal2021/dr22p.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/dr22p.pas
## dr22p.pas(101,3) Note: Local variable "t1" not used
## dr22p.pas(101,7) Note: Local variable "t2" not used
## dr22p.pas(145,6) Note: Local variable "j" not used
## dr22p.pas(150,3) Note: Local variable "s" not used
## Linking ../Pascal2021/dr22p
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 319 lines compiled, 0.2 sec
## 4 note(s) issued
```

```
Function: Rosenbrock Banana Valley
Classical starting point (-1.2,1)
alg22.pas -- Nash Algorithm 22 version 2 1988-03-24
  Conjugate gradients function minimiser
Steplength saving factor multiplies best steplength found at the
  end of each iteration as a starting value for next search
Enter a steplength saving factor (sugg. 1.7) -- setstep  1.7000000000000000E+000
Choose method (1=FR, 2=PR, 3=BS) 2
Method: Polak Ribiere
tolerance used in gradient test= 1.9999999999999998E-021
Exiting from Alg22.pas conjugate gradients minimiser
    147 function evaluations used
```

```
   35 gradient evaluations used

 Minimum function value found = 1.8188767421331468E-022
 At parameters
 Bvec[1]= 9.9999999998652422E-001
 Bvec[2]= 9.9999999997299449E-001
```

R

In the early 2000s, one of us (JN) tried implementing his optimization algorithms from Nashlib in R. The outcome of this exercise, coloured by collaborations with other workers (in particular Ravi Varadhan, Hans Wener Borchers, Ben Bolker, Duncan Murdoch, and Gabor Grothendiek), is the R-project package `optimx`. However, in considering Algorithm 22, which which I have never been happy, I came across Dai and Yuan (1999), where a very small change in the logic of Algorithm 22 allowed the three search direction updates offered separately in Algorithm 22 to be very elegantly combined. The resulting program, first packaged as R package `Rcgmin` before it was subsumed into `optimx` in 2019, works much, much better than the original Algorithm 22.

Algorithm 23 – Marquardt method for nonlinear least squares

Fortran

Listing

```
      SUBROUTINE A23MRT(N,B,M,TOL,A,C,N2,X,V,D,RES,DRES,NOCOM,P0,IFN,
     #IG,F,IPR)
C  ALGORITHM 23  MODIFIED MARQUARDT NONLINEAR SUM OF SQUARES
C    MINIMISATION
C  J.C. NASH   JULY 1978, FEBRUARY 1980, APRIL 1989
C     N  =  NO. OF PARAMETERS TO BE ADJUSTED
C     B  =  INITIAL POINT (SET OF PARAMETERS
C     M  =  NO. OF RESIDUALS
C    TOL =  RESET VALUE FOR MARQUARDT PARAMETER LAMBDA
C    A.C =  WORKING VECTORS OF N2 ELEMENTS
C  X,V,D =  WORKING VECTORS OF N  ELEMENTS
C  RES   =  NAME OF FUNCTION TO CALCULATE RESIDUAL NO. I
C           RVAL=RES(N,B,I,NOCOM)
C  DRES  =  NAME OF SUBROUTINE TO CALCULATE DERIVATIVES OF RESIDUAL I
C           CALL DRES(N,B,I,D)
C  NOCOM =  LOGICAL FLAG SET .TRUE. IF INITIAL POINT INFEASIBLE
C  P0    =  MINIMAL VALUE OF SUM OF SQUARES (OUTPUT)
C  IFN   = LIMIT ON FUNCTION EVALUATIONS (INPUT)  (SUM OF SQUARES)
C        =  COUNT OF FUNCTION EVALUATIONS (OUTPUT)
C  IG    =  COUNT OF DERIVATIVE EVALUATIONS
C  F     =  WORKING VECTOR OF LENGTH M USED TO SAVE RESIDUALS
C  IPR   = PRINT CHANNEL  IPR.GT.0 FOR PRINTING.
C  STEP 0
      LOGICAL NOCOM
      INTEGER N,M,N2,IFN,IG,LIM,I,J,Q,IJ,J1,COUNT
      REAL B(N),X(N),V(N),D(N),A(N2),C(N2),F(M)
      REAL  S,TOL,INC,DEC,LAMBDA,PHI,P,P0
C  FOR SAFETY RESET N2
      N2=N*(N+1)/2
```

```
C  PHI - NASH ADDITION TO MARQUARDT ALGORITHM
      PHI=1.0
C  INCREASE AND DECREASE FACTORS
      INC=10.0
      DEC=0.4
      LIM=IFN
      IFN=0
      IG=0
      LAMBDA=TOL
C  STEP 1
      P=0.0
C  BETTER DONE DOUBLE PRECISION
      IFN=IFN+1
      DO 15 I=1,M
        F(I)=RES(N,B,I,NOCOM)
        IF(NOCOM)RETURN
        P=P+F(I)**2
  15  CONTINUE
C  STEP 2
  20  IG=IG+1
      LAMBDA=LAMBDA*DEC
      P0=P
      IF(IPR.GT.0)WRITE(IPR,959)IG,IFN,P0
 959  FORMAT( 6H ITN #,I4, 8H EVALN *,I4,13H  SUMSQUARES=,1PE16.8)
C  STEP 3
      DO 34 J=1,N2
        A(J)=0.0
  34  CONTINUE
      DO 36 J=1,N
        V(J)=0.0
  36  CONTINUE
C  STEP 4
      DO 48 I=1,M
        CALL DRES(N,B,I,D)
        S=F(I)
        DO 46 J=1,N
           V(J)=V(J)+S*D(J)
           Q=J*(J-1)/2
           DO 44 K=1,J
            IJ=Q+K
             A(IJ)=A(IJ)+D(J)*D(K)
  44       CONTINUE
  46    CONTINUE
  48  CONTINUE
C  STEP 5
      DO 54 J=1,N2
        C(J)=A(J)
  54  CONTINUE
      DO 56 J=1,N
        D(J)=B(J)
  56  CONTINUE
C  STEP 6
  60  DO 68 J=1,N
```

```
        Q=J*(J+1)/2
        A(Q)=C(Q)*(1.0+LAMBDA)+PHI*LAMBDA
        X(J)=-V(J)
        IF(J.EQ.1)GOTO 68
        J1=J-1
        DO 66 I=1,J1
          IJ=Q-I
          A(IJ)=C(IJ)
  66    CONTINUE
  68  CONTINUE
C  STEP 7
      NOCOM=.FALSE.
      CALL A7CH(A,N2,N,NOCOM)
      IF(NOCOM)GOTO 130
C  STEP 8
      CALL A8CS(A,N2,X,N)
C  STEP 9
      COUNT=0
      DO 95 I=1,N
        B(I)=D(I)+X(I)
        IF(B(I).EQ.D(I))COUNT=COUNT+1
  95  CONTINUE
C  STEP 10
      IF(COUNT.EQ.N)RETURN
C  STEP 11
      IFN=IFN+1
      IF(IFN.GT.LIM)GOTO 140
      NOCOM=.FALSE.
      P=0.0
      DO 115 I=1,M
        F(I)=RES(N,B,I,NOCOM)
        IF(NOCOM)GOTO 130
        P=P+F(I)**2
 115  CONTINUE
C  STEP 12
      IF(P.LT.P0)GOTO 20
C  STEP 13
 130  LAMBDA=LAMBDA*INC
      IF(LAMBDA.EQ.0.0)LAMBDA=TOL
      GOTO 60
C  RESET PARAMETERS
 140  DO 144 I=1,N
        B(I)=D(I)
 144  CONTINUE
      RETURN
      END
```

Example output

We again test the code with a rather nasty 4-parameter problem called WOOD4 (Nash and Walker-Smith (1987), page 421) from different starting points. An example is included with a function evaluation limit that forces early termination. Caution!!

```
gfortran ../fortran/a23.f
mv ./a.out ../fortran/a23f.run
../fortran/a23f.run < ../fortran/dr23f.in
```

```
##  PROBLEM WOOD4  FN EVAL LIMIT=  300  LAMBDA TOL=  0.10000000E+01
##         -3.00000       -1.00000       -3.00000       -1.00000
##  CONV. TO  0.00000000E+00 IN   56 FNS &  44 DERS
##          1.00000        1.00000        1.00000        1.00000
##
##  PROBLEM WOOD4  FN EVAL LIMIT=  300  LAMBDA TOL=  0.10000000E+00
##         -3.00000       -1.00000       -3.00000       -1.00000
##  CONV. TO  0.00000000E+00 IN   57 FNS &  44 DERS
##          1.00000        1.00000        1.00000        1.00000
##
##  PROBLEM WOOD4  FN EVAL LIMIT=  300  LAMBDA TOL=  0.99999997E-04
##         -3.00000       -1.00000       -3.00000       -1.00000
##  CONV. TO  0.00000000E+00 IN   91 FNS &  66 DERS
##          1.00000        1.00000        1.00000        1.00000
##
##  PROBLEM WOOD4  FN EVAL LIMIT=  300  LAMBDA TOL=  0.10000000E-06
##         -3.00000       -1.00000       -3.00000       -1.00000
##  CONV. TO  0.00000000E+00 IN  101 FNS &  71 DERS
##          1.00000        1.00000        1.00000        1.00000
##
##  PROBLEM WOOD4  FN EVAL LIMIT=   10  LAMBDA TOL=  0.99999997E-04
##         -3.00000       -1.00000       -3.00000       -1.00000
##  CONV. TO  7.87695885E+00 IN   11 FNS &  10 DERS
##         -0.97191        0.95476       -0.96557        0.94363
##
##  PROBLEM WOOD4  FN EVAL LIMIT=    0  LAMBDA TOL=  0.00000000E+00
```

BASIC

Listing

Example output

Pascal

Listing

```
procedure modmrt( n : integer;
          var Bvec : rvector;
          var X : rvector;
          var Fmin  : real;
            Workdata : probdata);

{modified 1991 - 01 - 13}
var
  a, c: smatvec;
  delta, v : rvector;
  dec, eps, inc, lambda, p, phi, res : real;
  count, i, ifn, igrad, j, k, nn2, q : integer;
  notcomp, singmat, calcmat: boolean;
```

```
begin
  writeln('alg23.pas -- Nash Marquardt nonlinear least squares');
  with Workdata do
  begin
    if nlls = false then halt;
    Fmin:=big;
    inc:=10.0;
    dec:=0.4;
    eps:=calceps;
    lambda:=0.0001;
    phi:=1.0;
    ifn:=0;  igrad:=0;
    calcmat:=true;
    nn2:=(n*(n+1)) div 2;
    p:=0.0;
    for i:=1 to m do
    begin
      res:=nlres(i, n, Bvec, notcomp, Workdata);

      if notcomp then halt;
      p:=p+res*res;
    end;
    ifn:=ifn+1;
    Fmin:=p;
    count:=0;

    while count<n do
    begin

      if calcmat then
      begin
        writeln(igrad,' ',ifn,'  sum of squares=',Fmin);
        for i:=1 to n do
        begin
          write(Bvec[i]:10:5,' ');
          if (7 * (i div 7) = i) and (i<n) then writeln;
        end;
        writeln;
        igrad:=igrad+1;
        for j:=1 to nn2 do a[j]:=0.0;
        for j:=1 to n do v[j]:=0.0;
        for i:=1 to m do
        begin
          nljac(i, n, Bvec, X, workdata);
          res:=nlres(i, n, Bvec, notcomp, Workdata);
          for j:=1 to n do
          begin
            v[j]:=v[j]+X[j]*res;
            q:=(j*(j-1)) div 2;
            for k:=1 to j do a[q+k]:=a[q+k]+X[j]*X[k];
          end;
        end;
        for j:=1 to nn2 do c[j]:=a[j];
```

```
        for j:=1 to n do X[j]:=Bvec[j];
      end;
      writeln('LAMDA =',lambda:8);
      for j:=1 to n do
      begin
        q:=(j*(j+1)) div 2;
        a[q]:=c[q]*(1.0+lambda)+phi*lambda;
        delta[j]:=-v[j];
        if j>1 then
          for i:=1 to (j-1) do a[q-i]:=c[q-i];
      end;
      notcomp:=false;
      Choldcmp(n, a, singmat);
      if (not singmat) then
      begin
        Cholback(n, a, delta);
        count:=0;
        for i:=1 to n do
        begin
          Bvec[i]:=X[i]+delta[i];
          if (reltest + Bvec[i])=(reltest+X[i]) then count:=count+1;
        end;
        if count<n then
        begin
          p:=0.0; i:=0;
          repeat
            i:=i+1; res:=nlres(i,n,Bvec,notcomp, Workdata);
            if (not notcomp) then  p:=p+res*res;
          until notcomp or (i>=m);  {MODIFICATION m replaces n 1991-01-13}
          ifn:=ifn+1;
        end;
      end;
      if count<n then
        if (not singmat) and (not notcomp) and (p<Fmin) then
        begin
          lambda:=lambda*dec;
          Fmin:=p;
          calcmat:=true;
        end
      else
      begin
        lambda:=lambda*inc;
        if lambda<eps*eps then lambda:=eps;
        calcmat:=false;
      end;
    end;
  end;
end;
```

Example output

```
fpc ../Pascal2021/dr23p.pas
# copy to run file
```

```
mv ../Pascal2021/dr23p ../Pascal2021/dr23p.run
../Pascal2021/dr23p.run  >../Pascal2021/dr23p.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/dr23p.pas
## dr23p.pas(108,3) Note: Local variable "t1" not used
## dr23p.pas(108,7) Note: Local variable "t2" not used
## dr23p.pas(326,1) Note: Local variable "fail" not used
## dr23p.pas(327,1) Note: Local variable "mytol" is assigned but never used
## Linking ../Pascal2021/dr23p
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 345 lines compiled, 0.2 sec
## 4 note(s) issued
```

```
Function: Rosenbrock Banana Valley
Classical starting point (-1.2,1)
alg23.pas -- Nash Marquardt nonlinear least squares
0 1  sum of squares= 2.4199999999999996E+001
  -1.20000    1.00000
LAMDA = 1.0E-004
LAMDA = 1.0E-003
LAMDA = 1.0E-002
1 4  sum of squares= 4.1951264797246708E+000
  -0.94035    0.81867
LAMDA = 4.0E-003
LAMDA = 4.0E-002
2 6  sum of squares= 3.4489331941762451E+000
  -0.85681    0.73071
LAMDA = 1.6E-002
3 7  sum of squares= 2.8884746482625707E+000
  -0.67738    0.43149
LAMDA = 6.4E-003
LAMDA = 6.4E-002
4 9  sum of squares= 2.5543697866190191E+000
  -0.59818    0.35640
LAMDA = 2.6E-002
5 10  sum of squares= 2.0737924376402606E+000
  -0.40563    0.13324
LAMDA = 1.0E-002
LAMDA = 1.0E-001
6 12  sum of squares= 1.6678031989584301E+000
  -0.28917    0.07598
LAMDA = 4.1E-002
LAMDA = 4.1E-001
7 14  sum of squares= 1.5222760028883247E+000
  -0.23312    0.05845
LAMDA = 1.6E-001
8 15  sum of squares= 1.1856074935901313E+000
  -0.08192   -0.00556
LAMDA = 6.6E-002
LAMDA = 6.6E-001
9 17  sum of squares= 9.0261560461464696E-001
```

```
   0.14418   -0.02047
LAMDA = 2.6E-001
10 18  sum of squares= 6.3128696085431346E-001
   0.25563    0.03756
LAMDA = 1.0E-001
11 19  sum of squares= 4.4876187272348511E-001
   0.34989    0.10626
LAMDA = 4.2E-002
12 20  sum of squares= 3.1533416890084870E-001
   0.46770    0.20086
LAMDA = 1.7E-002
13 21  sum of squares= 1.9821770336917410E-001
   0.59510    0.33563
LAMDA = 6.7E-003
14 22  sum of squares= 1.0991718935771065E-001
   0.72879    0.51206
LAMDA = 2.7E-003
15 23  sum of squares= 4.6335934033970697E-002
   0.85143    0.70936
LAMDA = 1.1E-003
16 24  sum of squares= 1.0202905934304653E-002
   0.94094    0.87718
LAMDA = 4.3E-004
17 25  sum of squares= 6.2137787243781697E-004
   0.98563    0.96944
LAMDA = 1.7E-004
18 26  sum of squares= 5.7871308416976625E-006
   0.99824    0.99631
LAMDA = 6.9E-005
19 27  sum of squares= 9.8420017417796423E-009
   0.99991    0.99981
LAMDA = 2.7E-005
20 28  sum of squares= 4.2765767515536044E-012
   1.00000    1.00000
LAMDA = 1.1E-005
21 29  sum of squares= 3.2987189938435626E-016
   1.00000    1.00000
LAMDA = 4.4E-006
22 30  sum of squares= 4.1210148871538129E-021
   1.00000    1.00000
LAMDA = 1.8E-006
23 31  sum of squares= 8.2726980313912423E-027
   1.00000    1.00000
LAMDA = 7.0E-007
24 32  sum of squares= 1.2325951644078309E-030
   1.00000    1.00000
LAMDA = 2.8E-007

 Minimum function value found = 1.2325951644078309E-030
 At parameters
 Bvec[1]= 1.0000000000000000E+000
 Bvec[2]= 9.9999999999999989E-001
```

Algorithm 27 – Hooke and Jeeves pattern search minimization

BASIC

The Hooke and Jeeves code below is a modified version of that in Nash and Walker-Smith (1990).

Listing

```
40 DIM B(25),X(25)
50 LET N=2
60 LET B(1)=-1.2
70 LET B(2)=1
72 LET S1=0
74 LET S2=0
80 GOSUB 1000
90 GOSUB 1488: REM PRINT PARAMETERS
200 SYSTEM
1000 PRINT "Hooke and Jeeves -- 19851018, 19880809"
1008 REM CALLS:
1012 REM     FUNCTION F(B)  -- line 2000
1016 REM     ENVRON (computing  environment) -- line 7120
1020 REM
1024 REM INPUTS TO THE ROUTINE:
1028 REM    B() -- a vector of initial parameter estimates
1032 REM    S1  -- initial stepsize for search (set to 1 if zero)
1036 REM    S2  -- stepsize reduction factor, which is applied to
1040 REM           S1 when axial search fails (set to 0.1 if zero)
1044 REM    N   -- the number of parameters in the function F(B)
1060 REM OUTPUT FROM THE ROUTINE:
1064 REM    X() -- a vector of final parameter estimates for the
1068 REM           values of the parameters which minimize the function.
1072 REM    F0  -- value of the function at the minimum
1076 REM    I8  -- number of gradient evaluations (unchanged)
1080 REM    I9  -- number of function evaluations
1084 REM
1088 IF S1<=0 THEN LET S1=1: REM !! warning -- is variable undefined?
1092 IF S2<=0 THEN LET S2=.1: REM !!  ditto
1096 LET J8=1: REM no of fn eval before parameter display - mod 19880809
1100 LET J7=1: REM counter for parameter display
1104 GOSUB 7120: REM computing environment
1108 GOSUB 1440: REM copy B() into X() (lowest point so far)
1112 PRINT "STEP-SIZE =";S1
1116 PRINT "STEP-SIZE REDUCTION FACTOR =";S2
1128 GOSUB 1456: REM compute function in F, set I3<>0 if not possible.
1132 IF I3<>0 THEN 1412
1136 PRINT "INITIAL FUNCTION VALUE =";F
1144 LET F1=F: REM store function value at base point
1148 LET F0=F: REM store lowest function value so far
1152 GOSUB 1252: REM axial exploratory search
1156 IF I6=2*N THEN 1176: REM parameters unchanged in axial search
1160 IF F0>=F1 THEN 1176: REM test for a lower function value
1164 LET F1=F0: REM update function value at base
1168 GOSUB 1368: REM pattern move
1172 GOTO 1152: REM repeat axial search
```

```
1176 FOR J=1 TO N: REM is B() still the current base point?
1180 IF B(J)<>X(J) THEN 1192: REM test for changes in parameters
1184 NEXT J: REM in above test look for equality since B:=X at base
1188 GOTO 1208: REM reduce step-size as search has not reduced function
1192 GOSUB 1424: REM copy X into B (B() is now at the base point)
1196 PRINT " RETURN TO BASE POINT ";
1204 GOTO 1152: REM try another axial search
1208 LET S1=S1*S2: REM reduce step-size
1212 REM PRINT
1216 PRINT I9;F0;"STEPSIZE=";S1
1228 GOSUB 1476: REM display parameters
1232 IF I6<2*N THEN 1152: REM convergence test (no altered params)
1236 REM PRINT
1244 RETURN: REM function minimization complete
1248 REM axial exploratory search subroutine
1252 LET I6=0: REM counter for number of unchanged parameters
1256 FOR J=1 TO N
1264 LET S3=B(J): REM store parameter value
1272 LET B(J)=S3+S1: REM step forward
1280 IF B(J)+E5<>S3+E5 THEN 1292: REM test equality relative to E5
1284 LET I6=I6+1
1288 GOTO 1300: REM now try negative step
1292 GOSUB 1456: REM function evaluation
1296 IF F<F0 THEN 1340: REM test if function value < current lowest value
1300 LET B(J)=S3-S1: REM step backward
1312 IF B(J)+E5=S3+E5 THEN 1328: REM test equality
1316 GOSUB 1456: REM function evaluation
1320 IF F<F0 THEN 1340
1324 GOTO 1332
1328 LET I6=I6+1: REM count number of times parameter unchanged
1332 LET B(J)=S3: REM restore original parameter (not by addition!!)
1336 GOTO 1344
1340 LET F0=F: REM store new lowest function value
1344 NEXT J
1348 REM PRINT " AXIAL SEARCH F0=";F0
1356 REM GOSUB 1476: REM print parameters
1360 RETURN: REM end axial search
1364 REM PATTERN MOVE
1368 FOR J=1 TO N
1376 LET S3=2*B(J)-X(J): REM element of new base point
1380 LET X(J)=B(J): REM store current point
1392 LET B(J)=S3: REM store new base point
1396 NEXT J
1400 REM PRINT " PMOVE ";
1408 RETURN: REM end pattern move
1412 PRINT "FUNCTION NOT COMPUTABLE AT INITIAL POINT"
1420 STOP
1424 FOR J=1 TO N: REM copy X into B
1428 LET B(J)=X(J)
1432 NEXT J
1436 RETURN
1440 FOR J=1 TO N: REM copy B into X
1444 LET X(J)=B(J)
```

```
1448 NEXT J
1452 RETURN
1456 LET I3=0: REM compute function -- reset failure flag
1460 GOSUB 2000: REM user routine
1464 IF I3<>0 THEN LET F=B9: REM large value assigned
1468 LET I9=I9+1: REM function evaluation counter
1472 RETURN
1476 IF J8=0 THEN RETURN: REM no parameter display
1480 IF I9<J7*J8 THEN RETURN: REM check if to be printed
1484 LET J7=INT(I9/J8)+1: REM parameter display control
1488 PRINT "parameters";
1496 FOR J=1 TO N
1500 LET Q$=""
1516 PRINT "  ";B(J);Q$;
1524 IF 5*INT(J/5)<>J THEN 1544
1528 PRINT
1532 PRINT "  ";
1544 NEXT J
1548 PRINT
1556 RETURN
2000 I3=0: REM FUNCTION IS COMPUTABLE
2010 LET F=((B(2)-B(1)^2)^2)*100.0+(1.0-B(1))^2
2020 RETURN
7120 LET E9=2^(-52): REM MACHINE EPSILON -- PRE-COMPUTED HERE
7130 LET E5=10000: REM RELATIVE SHIFT FOR COMPARISONS
7140 LET B9=1E35: REM A BIG NUMBER
7150 RETURN
```

Example output

```
bas ../BASIC/hj27.bas <../BASIC/yes.in
```

```
## Hooke and Jeeves -- 19851018, 19880809
## STEP-SIZE = 1
## STEP-SIZE REDUCTION FACTOR = 0.1
## INITIAL FUNCTION VALUE = 24.2
##  5  24.2 STEPSIZE= 0.1
## parameters  -1.2   1
##  RETURN TO BASE POINT  19  4.42 STEPSIZE= 0.01
## parameters  -1.1   1.2
##  RETURN TO BASE POINT  RETURN TO BASE POINT  RETURN TO BASE POINT  162
##  0.007696 STEPSIZE= 0.001
## parameters   0.92   0.85
##  RETURN TO BASE POINT  177  0.006247 STEPSIZE= 0.0001
## parameters   0.921   0.848
##  RETURN TO BASE POINT  RETURN TO BASE POINT  322  1.523313e-05 STEPSIZE= 1e-05
## parameters   1.0039   1.0078
##  RETURN TO BASE POINT  923  3.600001e-09 STEPSIZE= 1e-06
## parameters   1.00006   1.00012
##  927  3.600001e-09 STEPSIZE= 1e-07
## parameters   1.00006   1.00012
##  RETURN TO BASE POINT  1996  6.399999e-13 STEPSIZE= 1e-08
## parameters   1.000001   1.000002
```

```
##  2000  6.399999e-13 STEPSIZE= 1e-09
## parameters   1.000001    1.000002
##  RETURN TO BASE POINT  3488  4.899814e-17 STEPSIZE= 1e-10
## parameters   1    1
##  3492  4.899814e-17 STEPSIZE= 1e-11
## parameters   1    1
##  RETURN TO BASE POINT  4770  8.07604e-21 STEPSIZE= 1e-12
## parameters   1    1
##  4772  8.07604e-21 STEPSIZE= 1e-13
## parameters   1    1
##  4772  8.07604e-21 STEPSIZE= 1e-14
## parameters   1    1
## Quit without saving? (y/n) y
```

Pascal

Listing

```
procedure hjmin(n: integer;
        var B,X: rvector;
        var Fmin: real;
            Workdata: probdata;
        var fail: boolean;
            intol: real);

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

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

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

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

  fval := fminfn(n, B,Workdata,notcomp);
  if notcomp then
  begin
    writeln('*** FAILURE *** Function not computable at initial point');
    fail := true;
  end
  else
```

```
begin
  writeln('Initial function value =',fval);
  for i := 1 to n do
  begin
    write(B[i]:10:5,' ');
    if (7 * (i div 7) = i) and (i<n) then writeln;
  end;
  writeln;
  fold := fval; Fmin := fval;
  while stepsize>intol do
  begin

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

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

        write('stepsize now ',stepsize:10,'  Best fn value=',Fmin);
```

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

Example output

Example output

Use Rosenbrock banana-shaped valley problem in 2 dimensions.

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

```
## Free Pascal Compiler version 3.0.4+dfsg-23 [2019/11/25] for x86_64
## Copyright (c) 1993-2017 by Florian Klaempfl and others
## Target OS: Linux for x86-64
## Compiling ../Pascal2021/dr27.pas
## Linking ../Pascal2021/dr27
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 303 lines compiled, 0.2 sec
```

```
Function: Rosenbrock Banana Valley
Classical starting point (-1.2,1)
Initial function value = 2.4199999999999996E+001
  -1.20000    1.00000
Return to old base point
stepsize now  4.80E-002  Best fn value= 4.4562559999999998E+000 after 12
  -0.96000    1.00000
Return to old base point
stepsize now  9.60E-003  Best fn value= 4.0578692096000006E+000 after 24
  -1.00800    1.00000
Return to old base point
Return to old base point
Return to old base point
stepsize now  1.92E-003  Best fn value= 1.0707319193525812E-003 after 161
   0.98880    0.98080
Return to old base point
stepsize now  3.84E-004  Best fn value= 1.3884319308353293E-004 after 172
```

```
   0.99072    0.98080
Return to old base point
stepsize now  7.68E-005  Best fn value= 9.3512661164573335E-005 after 184
   0.99034    0.98080
Return to old base point
stepsize now  1.54E-005  Best fn value= 1.1312841503153136E-007 after 387
   0.99978    0.99954
Return to old base point
stepsize now  3.07E-006  Best fn value= 5.6836523263813657E-008 after 399
   0.99977    0.99954
Return to old base point
stepsize now  6.14E-007  Best fn value= 5.2964452813167824E-008 after 410
   0.99977    0.99954
Return to old base point
stepsize now  1.23E-007  Best fn value= 1.4270706145809712E-011 after 506
   1.00000    0.99999
Return to old base point
stepsize now  2.46E-008  Best fn value= 9.4796228739601164E-012 after 517
   1.00000    0.99999
Return to old base point
stepsize now  4.92E-009  Best fn value= 9.4690619892340589E-012 after 529
   1.00000    0.99999
Return to old base point
Return to old base point
stepsize now  9.83E-010  Best fn value= 4.4567065650768205E-015 after 661
   1.00000    1.00000
Return to old base point
stepsize now  1.97E-010  Best fn value= 4.0727302462025689E-015 after 673
   1.00000    1.00000
Return to old base point
stepsize now  3.93E-011  Best fn value= 5.5957365459244406E-019 after 1084
   1.00000    1.00000
Return to old base point
stepsize now  7.86E-012  Best fn value= 1.7697158812184515E-019 after 1096
   1.00000    1.00000
Return to old base point
stepsize now  1.57E-012  Best fn value= 1.5428882521329409E-019 after 1107
   1.00000    1.00000
Return to old base point
stepsize now  3.15E-013  Best fn value= 2.9714142551459652E-023 after 1190
   1.00000    1.00000
Return to old base point
stepsize now  6.29E-014  Best fn value= 3.6918757417520296E-024 after 1201
   1.00000    1.00000
Return to old base point
stepsize now  1.26E-014  Best fn value= 2.5495050765906063E-024 after 1213
   1.00000    1.00000
Return to old base point
stepsize now  2.52E-015  Best fn value= 4.7610344079933319E-027 after 1293
   1.00000    1.00000
Return to old base point
stepsize now  5.03E-016  Best fn value= 3.9955928108960689E-027 after 1304
   1.00000    1.00000
```

```
Converged to Fmin= 3.9955928108960689E-027 after 1304 evaluations

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

R

Listing

```r
# hjn.R -- R implementation of J Nash BASIC HJG.BAS 20160705
hjn <- function(par, fn, lower=-Inf, upper=Inf, bdmsk=NULL, control=list(trace=0), ...){
  n <- length(par) # number of parameters
  if (! is.null(control$maximize) && control$maximize)
           stop("Do NOT try to maximize with hjn()")
  if (is.null(control$trace)) control$trace <- 0 # just in case
  if (is.null(control$stepsize)) {
     stepsize <- 1 # initial step size (could put in control())
  } else { stepsize <- control$stepsize }
  # Want stepsize positive or bounds get messed up
  if (is.null(control$stepredn)) {
     stepredn <- .1 # defined step reduction (again in control()??)
  } else { stepredn <- control$stepredn }
  if (is.null(control$maxfeval)) control$maxfeval<-2000*n
  if (is.null(control$eps)) control$epsl<-1e-07
  steptol <- control$eps
  # Hooke and Jeeves with bounds and masks
  if (length(upper) == 1) upper <- rep(upper, n)
  if (length(lower) == 1) lower <- rep(lower, n)
  if (is.null(bdmsk)) {
      bdmsk <- rep(1,n)
      idx <- 1:n
  } else { idx <- which(bdmsk != 0) } # define masks
  if (any(lower >= upper)){
      warning("hjn: lower >= upper for some parameters -- set masks")
      bdmsk[which(lower >= upper)] <- 0
      idx <- which(bdmsk != 0)
  }
  if (control$trace > 0) {
    cat("hjn:bdmsk:")
    print(bdmsk)
#   cat("idx:")
#   print(idx)
  }
  nac <- length(idx)
  offset = 100. # get from control() -- used for equality check
  if (any(par < lower) || any(par > upper)) stop("hjn: initial parameters out of bounds")
  pbase <- par # base parameter set (fold is function value)
  f <- fn(par, ...) # fn at base point
  fmin <- fold <- f # "best" function so far
  pbest <- par # Not really needed
  fcount <- 1 # count function evaluations, compare with maxfeval
```

```
#     cat(fcount, "  f=",fold," at ")
#     print(par)
#     tmp <- readline("cont.")
  keepgoing <- TRUE
  ccode <- 1 # start assuming won't get to solution before feval limit
  while (keepgoing) {
    # exploratory search -- fold stays same for whole set of axes
    if (control$trace > 0) cat("Exploratory move - stepsize = ",stepsize,"\n")
    if (control$trace > 1) {
       cat("p-start:")
       print(par)
    }
    for (jj in 1:nac){ # possibly could do this better in R
       # use unmasked parameters
       j <- idx[jj]
       ptmp <- par[j]
       doneg <- TRUE # assume we will do negative step
       if (ptmp + offset < upper[j] + offset) { # Not on upper bound so do pos step
          par[j] <- min(ptmp+stepsize, upper[j])
          if ((par[j] + offset) != (ptmp + offset)) {
             fcount <- fcount + 1
             f <- fn(par, ...)
#               cat(fcount, "  f=",f," at ")
#               print(par)
             if (f < fmin) {
                fmin <- f
                pbest <- par
#                 cat("*")
                doneg <- FALSE # only case where we don't do neg
                resetpar <- FALSE
             }
#             tmp <- readline("cont>")
          }
       } # end not on upper bound
       if (fcount >= control$maxfeval) break
       if (doneg) {
         resetpar <- TRUE # going to reset the paramter unless we improve
         if ((ptmp + offset) > (lower[j] + offset)) { # can do negative step
            par[j] <- max((ptmp - stepsize), lower[j])
            if ((par[j] + offset) != (ptmp + offset)) {
               fcount <- fcount + 1
               f <- fn(par, ...)
#                 cat(fcount, "  f=",f," at ")
#                 print(par)
               if (f < fmin) {
                  fmin <- f
                  pbest <- par
#                  cat("*")
                  resetpar <- FALSE # don't reset parameter
               }
#               tmp <- readline("cont<")
            }
         } #  neg step possible
```

```
      } # end doneg
      if (resetpar) { par[j] <- ptmp }
   } # end loop on axes
   if (fcount >= control$maxfeval) {
       ccode <- 1
       if (control$trace > 0) cat("Function count limit exceeded\n")
       ans <- list(par=pbest, value=fmin, counts=c(fcount, NA), convergence=ccode)
       return(ans)
   }
   if (control$trace > 0) {
      cat("axial search with stepsize =",stepsize,"  fn value = ",fmin,"  after ",fcount,"  maxfeval =
   }
   if (fmin < fold) { # success -- do pattern move (control$trace > 0) cat("Pattern move \n")
      if (control$trace > 1) {
         cat("PM from:")
         print(par)
         cat("pbest:")
         print(pbest)
      }
      for (jj in 1:nac) { # Note par is best set of parameters
         j <- idx[jj]
         ptmp <- 2.0*par[j] - pbase[j]
         if (ptmp > upper[j]) ptmp <- upper[j]
         if (ptmp < lower[j]) ptmp <- lower[j]
         pbase[j] <- par[j]
         par[j] <- ptmp
      }
      fold <- fmin
      if (control$trace > 1) {
         cat("PM to:")
         print(par)
      }
   # Addition to HJ -- test new  base
#       fcount <- fcount + 1
#       f <- fn(par, ...)
#         cat(fcount, "  f=",f," at ")
#         print(par)
#         tmp <- readline("PM point")
#       if (f < fmin) {
#         if (control$trace > 0) {cat("Use PM point as new base\n")}
#         pbest <- pbase <- par
#       }
   } else { # no success in Axial Seart, so reduce stepsize
      if (fcount >= control$maxfeval) {
       ccode <- 1
       if (control$trace > 0) cat("Function count limit exceeded\n")
       ans <- list(par=pbest, value=fmin, counts=c(fcount, NA), convergence=ccode)
       return(ans)
      }
      # first check if point changed
      samepoint <- identical((par + offset),(pbase + offset))
      if (samepoint) {
         stepsize <- stepsize*stepredn
```

```
         if (control$trace > 1) cat("Reducing step to ",stepsize,"\n")
         if (stepsize <= steptol) keepgoing <- FALSE
         ccode <- 0 # successful convergence
      } else { # return to old base point
         if (control$trace > 1) {
            cat("return to base at:")
            print(pbase)
         }
         par <- pbase
      }
   }
   if (fcount >= control$maxfeval) {
       ccode <- 1
       if (control$trace > 0) cat("Function count limit exceeded\n")
       ans <- list(par=pbest, value=fmin, counts=c(fcount, NA), convergence=ccode)
       return(ans)
   }
  } # end keepgoing loop
  if ( control$trace > 1 ) {
    if (identical(pbest, pbase)) {cat("pbase = pbest\n") }
    else { cat("BAD!: pbase != pbest\n") }
  }

  ans <- list(par=pbest, value=fmin, counts=c(fcount, NA), convergence=ccode)
}
```

Example output

We use the Rosenbrock banana-shaped valley problem in 2 dimensions.

```
source("../R/hjn.R") # bring the Hooke and Jeeves minimizer code into workspace
fminfn <-function(x){
    val<-((x[2]-x[1]^2)^2)*100.0+(1.0-x[1])^2
    val
}
x0<-c(-1.2,1)
cat("Check fn at c(-1.2,1)=",fminfn(x0),"\n")
```

```
## Check fn at c(-1.2,1)= 24.2
```

```
rslt <- hjn(par=x0, fn=fminfn, control=list(trace=0))
print(rslt)
```

```
## $par
## [1] 1.000001 1.000002
##
## $value
## [1] 6.399999e-13
##
## $counts
## [1] 1996   NA
##
## $convergence
## [1] 0
```

```
## /versioned/Nash-Compact-Numerical-Methods/fortran
```

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

References

Brent, R. 1973. *Algorithms for Minimization Without Derivatives*. Englewood Cliffs, NJ: Prentice-Hall.

Dai, Y. H., and Y. Yuan. 1999. "A Nonlinear Conjugate Gradient Method with a Strong Global Convergence Property." *SIAM Journal on Optimization* 10: 177–82.

Maechler, Martin. 2020. *Rmpfr: R MPFR - Multiple Precision Floating-Point Reliable*. https://CRAN.R-project.org/package=Rmpfr.

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

Nash, John C., and M. Walker-Smith. 1990. "NLEX: Examples and Software Extensions for Nonlinear Modeling and Robust Nonlinear Estimation." *The American Statistician* 44 (1): 55.

Nash, John C., and Mary Walker-Smith. 1987. *Nonlinear Parameter Estimation: An Integrated System in BASIC*. New York: Marcel Dekker.