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

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;
```

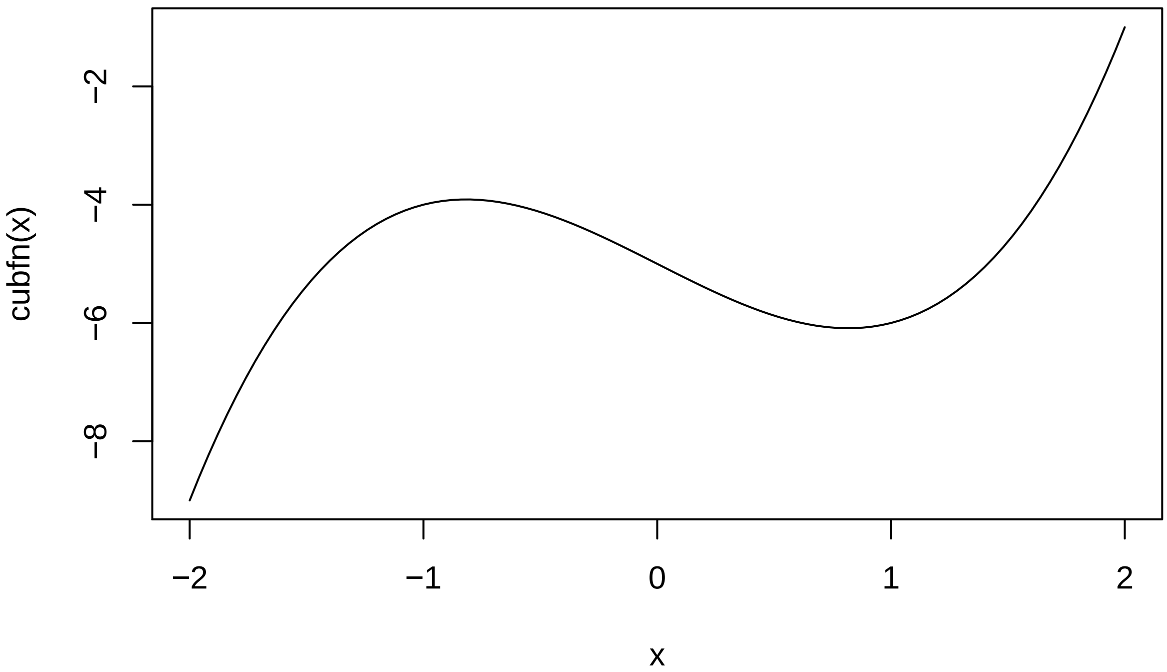
Example output

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)
```




```r
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

```fortran
      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

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, 0.1 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
```

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

```
## Tighter tolerance?
```

```r
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

```fortran
      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);
    write(confile,ifn,' evalns: f(',b:16,')=',fb:10);
    writeln('  width interval= ',(ubound-lbound):10);
    writeln(confile,'  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, 0.1 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
```

```
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
```

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.

Algorithm 27 – Hooke and Jeeves pattern search minimization

Pascal

Listing

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

  This program is designed to minimise functions of n parameters.

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

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

  Modified for Turbo Pascal 5.0

          Copyright 1988, 1990 J.C.Nash
}

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

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

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

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

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

(* remove the comments and delete the inclusion of ROSEN.PAS
   to use the JJACF.PAS test with EX27R.CNM
   {$I JJACF.PAS}
   Note that we move the inclusion to the right just in case.
*)

{rosen.pas
  == suite of procedures and functions defining the Rosenbrock
    banana shaped valley problem.
```

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

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

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

```
begin
  case i of
    1: begin
      jacrow[1]:=-20.0*Bvec[1];
      jacrow[2]:=10.0;
    end;
    2: begin
      jacrow[1]:=-1.0;
      jacrow[2]:=0.0;
    end;
    else halt; {safety stop}
  end; {case}
end; {nljac from rosen.pas}

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

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

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

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

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

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

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

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

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

      write('stepsize now ',stepsize:10,'  Best fn value=',Fmin);
      writeln(' after ',ifn);
      for i := 1 to n do
```

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

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

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

Example output

Use Rosenbrock banana-shaped valley problem in 2 dimensions.

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

```
## Free Pascal Compiler version 3.0.4+dfsg-23 [2019/11/25] for x86_64
```

```
## Copyright (c) 1993-2017 by Florian Klaempfl and others
## Target OS: Linux for x86-64
## Compiling ../Pascal2021/dr27.pas
## Linking ../Pascal2021/dr27
## /usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?
## 303 lines compiled, 0.1 sec
```

```
Function: Rosenbrock Banana Valley
Classical starting point (-1.2,1)
Initial function value = 2.4199999999999996E+001
  -1.20000    1.00000
Return to old base point
stepsize now  4.80E-002  Best fn value= 4.4562559999999998E+000 after 12
  -0.96000    1.00000
Return to old base point
stepsize now  9.60E-003  Best fn value= 4.0578692096000006E+000 after 24
  -1.00800    1.00000
Return to old base point
Return to old base point
Return to old base point
stepsize now  1.92E-003  Best fn value= 1.0707319193525812E-003 after 161
   0.98880    0.98080
Return to old base point
stepsize now  3.84E-004  Best fn value= 1.3884319308353293E-004 after 172
   0.99072    0.98080
Return to old base point
stepsize now  7.68E-005  Best fn value= 9.3512661164573335E-005 after 184
   0.99034    0.98080
Return to old base point
stepsize now  1.54E-005  Best fn value= 1.1312841503153136E-007 after 387
   0.99978    0.99954
Return to old base point
stepsize now  3.07E-006  Best fn value= 5.6836523263813657E-008 after 399
   0.99977    0.99954
Return to old base point
stepsize now  6.14E-007  Best fn value= 5.2964452813167824E-008 after 410
   0.99977    0.99954
Return to old base point
stepsize now  1.23E-007  Best fn value= 1.4270706145809712E-011 after 506
   1.00000    0.99999
Return to old base point
stepsize now  2.46E-008  Best fn value= 9.4796228739601164E-012 after 517
   1.00000    0.99999
Return to old base point
stepsize now  4.92E-009  Best fn value= 9.4690619892340589E-012 after 529
   1.00000    0.99999
Return to old base point
Return to old base point
stepsize now  9.83E-010  Best fn value= 4.4567065650768205E-015 after 661
   1.00000    1.00000
Return to old base point
stepsize now  1.97E-010  Best fn value= 4.0727302462025689E-015 after 673
   1.00000    1.00000
Return to old base point
```

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

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

Cleanup of working files

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

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

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

```
## /j19z/j19store/versioned/Nash-Compact-Numerical-Methods/Pascal2021
## /j19z/j19store/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.

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.