9.1 Find a Local Minimum of a Univariate Function

A. Purpose

Find a local minimum of a univariate function, $f(x)$, in the closed interval between specified abscissae, a and b, to within a specified tolerance, TOL.

B. Usage

This subroutine uses reverse communication, *i.e.*, it returns to the calling program each time it needs to have $f()$ evaluated at a new value of x.

B.1 Program Prototype, Single Precision

```
INTEGER  MODE
REAL  X, XORF, TOL

   MODE = 0
   X = An endpoint of the search interval
   XORF = The other endpoint
   TOL = Tolerance on x
10 continue

   CALL SFMIN(X, XORF, MODE, TOL)

   if( MODE .eq. 1) then
      XORF = f() evaluated at X
      go to 10
   endif
```

Computed quantities are returned in MODE, X, and XORF.

B.2 Argument Definitions

X, XORF, MODE [all are inout] When starting a new problem, the user must set MODE = 0, X = a, and XORF = b. The value of MODE should not be changed during the solution process, except to control detailed printing as explained in Section D.

a and b denote endpoints defining a closed interval in which a local minimum is to be found. Permit $a < b$ or $a > b$ or $a = b$.

On each return after a call with MODE = 0 or 1, this subroutine will set MODE to a value in the range [1:4] to indicate the action needed from the calling program or the status on termination.

- = 1 means the calling program must evaluate f(X), store the value in XORF, and then call this subroutine again.
- = 2 means normal termination. XORF contains the value of f(X) where X is a local minimum of the function in the interval specified.
- = 3 same as MODE = 2, except the requested accuracy was not obtained.
- = 4 means error termination due to SFMIN having been entered with MODE > 1.

TOL [in] An absolute tolerance on the uncertainty in the final estimate of the local minimum. If TOL $\leq$ 0, SFMIN attempts to get all the accuracy it can. MODE will not be set to 3 in this case. Let ϵ denote the machine precision, the value of which is obtained by reference to R1MACH(4) (D1MACH(4) for double precision), see Chapter 19.1. The operational tolerance, τ, at any trial abscissa, X, will be

$$\tau = (2/3) \times \text{TOL} + 2 \times |\text{X}| \times \epsilon^{1/2}.$$

B.3 Modifications for Double Precision

For double-precision usage change the name SFMIN to DFMIN, and change the REAL declaration to DOUBLE PRECISION.

C. Examples and Remarks

The program DRSFMIN illustrates the use of SFMIN to compute the local minimum of the function, $f(x) = 2^x + 2^{-2x}$, for which the exact answer is $x = 1/3$, with a minimum value of $(3/2)2^{1/3} \approx 1.88988$. Output is shown in ODSFMIN.

This algorithm evaluates f at an initial endpoint, a or b, only if the descent process leads to one of these points.

D. Functional Description

At the beginning of each iteration after the first, the algorithm has three ordered abscissae, $a' < x < b'$. The point x will have the smallest function value found up to this point. All further searching will be in the closed interval $[a', b']$. The algorithm will not necessarily have evaluated the function at a' and b'. There may also be up to two saved points, w and v, with associated saved function values satisfying $f(x) \leq f(w) \leq f(v)$. The points w and v may or may not coincide with a' and b'. If $\max(x - a',\ b' - x) \leq \tau$, where τ is the operational tolerance defined above in Section B, the algorithm terminates, returning x as the approximate abscissa of the local minimum.

If the algorithm has not converged then the algorithm generates a new point strictly between a' and b' by use of either

(1) parabolic interpolation using x, w, v, $f(x)$, $f(w)$, and $f(v)$, or

©1997 Calif. Inst. of Technology, 2015 Math à la Carte, Inc.

(2) golden section prediction using a', x, and b'.

The algorithm chooses between these two methods on the basis of tests designed to achieve a balance between rapid progress and reliability. Let e_i denote the distance of the i^{th} trial abscissa from the solution abscissa. For a sufficiently smooth function whose first derivative has a simple zero at the minimizing abscissa, and when the trial point is sufficiently close to the solution, Method 1 converges according to $e_{i+1} = e_i^{1.324}$. Method 2 is slower but more reliable, converging according to $(b' - a')_{i+1} = 0.618(b' - a')_i$ regardless of the smoothness of the function or the proximity to the solution. Note that five steps of Method 2 gains somewhat more than one decimal place in the solution since $(0.618)^5 \approx 0.090$.

This algorithm, with a Fortran 66 implementation, is presented in [1]. The algorithm is due to R. P. Brent, [2].

The algorithm as given in [1] never evaluates f at either of the initially given endpoints. In cases in which the minimum is at an endpoint, the algorithm returns a final abscissa that differs from the endpoint by τ, and may use many iterations. We have altered the code to permit evaluation at an endpoint when the search process approaches an endpoint, and to keep the number of iterations used for an endpoint minimum essentially the same as for an interior minimum.

Detailed printing

Before the initial call with MODE = 0, or any time during the iterative process, the user may initiate or stop the detailed output by calling SFMIN with a negative value of MODE. There is no problem-solving action on such a call. A saved counter is set so detailed output will be written using WRITE(*,...) on the next |MODE| − 1 normal calls. To resume normal computation the calling program must set MODE = 0 if a new problem sequence is being started or MODE = 1 if an iterative sequence is being continued.

The detailed output will consist of X, XORF, and IC. IC is an internal variable that is initially 0 and is incremented by 1 whenever a golden section move is used rather than a move determined from a parabola. If IC gets to 4, it will either be set to −99, or it serves as an internal flag to check the results at an endpoint.

References

1. G. E. Forsythe, M. A. Malcolm, and C. B. Moler, **Computer Methods for Mathematical Computations**, Prentice-Hall, Englewood Cliffs, N. J. (1977) 259 pages.
2. R. P. Brent, **Algorithms for Minimization Without Derivatives**, Prentice-Hall, Englewood Cliffs, N. J. (1973).

E. Error Procedures and Restrictions

SFMIN permits the given endpoints to satisfy $a < b$, $a > b$, or $a = b$. In the latter case the solution is $x = a$, and it will be found in one iteration.

The user must initially set MODE = 0 and must not alter MODE after that while iterating on the same problem, except to obtain the detailed printing as described in Section D. Entering SFMIN with MODE > 1 is an error condition and an error message will be issued using the error processing routines of Chapter 19.2 with an error level of 0.

SFMIN uses the Fortran 77 SAVE statement to save values of internal variables between the successive calls needed to solve a problem. Thus SFMIN cannot be used to work on more than one problem at a time. In particular, calling SFMIN with MODE = 0 will always initialize it for a new problem and no data will be retained from a previous problem, except for the internal detailed printing counter mentioned in Section D. If one needs this type of capability, possibly because of a recursive formulation of a multivariate minimization problem, one could make multiple copies of this subroutine, giving a distinct name to each copy. DMLC01, Chapter 9.2, is recommended for most such problems, however.

F. Supporting Information

The source language is ANSI Fortran 77.

Entry	Required Files
DFMIN	AMACH, DFMIN, ERFIN, ERMSG, IERM1, IERV1
SFMIN	AMACH, ERFIN, ERMSG, IERM1, IERV1, SFMIN

Adapted to Fortran 77 for the JPL MATH77 library from the subroutine, FMIN, in [1] by C. L. Lawson and F. T. Krogh, JPL, Oct. 1987.

DRSFMIN

```
c      program DRSFMIN
c>> 1996-05-28 DRSFMIN Krogh  Moved format up.
c>> 1994-10-19 DRSFMIN Krogh  Changes to use M77CON
c>> 1994-07-15 DRSFMIN CLL
c>> 1993-03-01 CLL Changed "zero" to "minimum".
c>> 1992-03-24 CLL Added missing comma in format statement.
c>> 1987-12-09 DRSFMIN  Lawson  Initial Code.
c      Demo driver for SFMIN.  Finds minimum of a univariate function.
c      C. L. Lawson & S. Y. Chiu, JPL, Aug 1987, Oct 1987
c      F. T. Krogh, Oct. 1987.
c     --------------------------------------------------------------------
c--S replaces "?": DR?FMIN, ?FMIN
c     --------------------------------------------------------------------
      real               A, B, X, XORF, TOL, XTOL, TWO
      parameter(A = -1.0E0, B = 1.0E0, XTOL = 1.0E-7)
      parameter(TWO = 2.0E0)
      integer MODE
    1 format(' Results returned by SFMIN:'/'     MODE = ',i3/
     *'     Solution:      X = ',f16.08/
     *'                  f(X) = ',f16.08)
c     --------------------------------------------------------------------
      write(*,'(1x,a)')
     * 'DRSFMIN.. Demo driver for SFMIN, univariate minimum finder.',
     * 'Problem:       Find minimum of 2**X + 2**(-2*X).',
     * 'Exact result: X = 1/3',
     * 'Min value: 1.5 * 2**(1/3) = 1.88988157...'
      X = A
      XORF = B
      TOL = XTOL
      MODE = 0
   10 call SFMIN(X, XORF, MODE, TOL)
      if (MODE .ne. 1) go to 20
         XORF = TWO**X + TWO**(-TWO*X)
      go to 10
   20 continue
      write(*,1) MODE, X, XORF
      stop
      end
```

ODSFMIN

```
DRSFMIN.. Demo driver for SFMIN, univariate minimum finder.
Problem:       Find minimum of 2**X + 2**(-2*X).
Exact result: X = 1/3
Min value: 1.5 * 2**(1/3) = 1.88988157...
Results returned by SFMIN:
   MODE =    3
    Solution:      X =        0.33340994
                 f(X) =        1.88988161
```