Module 9.5: nag_uv_min
Univariate Minimization

`nag_uv_min` provides a procedure for computing a minimum of a continuous function of a single variable in a given finite interval.

Contents

Introduction

This module contains the procedure `nag_uv_min_sol`, which attempts to find the location of a minimum of a function of a single variable over values of the variable in a given finite interval. If there is more than one minimum, `nag_uv_min_sol` will normally find one of them. The first derivative of the function need not be supplied by the user; however, the procedure is more reliable when the derivative is used.

Procedure: nag_uv_min_sol

1 Description

nag_uv_min_sol attempts to find a minimum of a continuous function of a single variable over values of the variable in a given finite interval $[a, b]$. The problem can be stated as follows:

minimize $f(x)$ subject to $a \leq x \leq b$.

It normally computes a sequence of x values which tend in the limit to a point x_{min} that is either a turning point or an endpoint corresponding to a minimum of $f(x)$ subject to the given bounds. It also progressively reduces the interval $[a, b]$ in which x_{min} is known to lie.

The method used is based on safeguarded polynomial approximation as described in Gill *et al.* [1]. Quadratic approximation is used unless you supply the first derivative of $f(x)$ (i.e., df/dx), in which case cubic approximation is used instead.

2 Usage

```
USE nag_uv_min
CALL nag_uv_min_sol(fun, a, b, x, f  [, optional arguments])
```

3 Arguments

3.1 Mandatory Arguments

fun — subroutine

The procedure fun, supplied by the user, must evaluate the objective function $f(x)$ and (optionally) its first derivative $g(x) = df/dx$ at a specified point x.

Its specification is:

```
subroutine fun(x,f,g)
```

```
real(kind=wp), intent(in) ::  x
```

Input: the point x at which the function f and (optionally) its first derivative df/dx are to be evaluated.

```
real(kind=wp), intent(out) ::  f
```

Output: the value of the function f at the point x.

```
real(kind=wp), intent(out), optional ::  g
```

Output: if present, g must contain the value of the first derivative df/dx at the point x.

Note: if the first derivative is not to be used in estimating x_{min} (i.e., the optional argument use_deriv (see Section 3.2) is set to .false.), then g will not be supplied in any call to fun; however, the definition of fun must still contain the argument g in its specification. (By default, fun will always be called with g present.)

Note: fun should be tested separately before being supplied to this procedure.

a — real(kind=*wp*), intent(inout)

b — real(kind=*wp*), intent(inout)

Input: a and b, the lower and the upper bounds of the interval containing x_{min}.

Output: improved lower and upper bounds of the interval containing the minimum.

Constraints: $\mathtt{b} > \mathtt{a} + \varepsilon_{abs} + \varepsilon_r|\mathtt{a}|$ (see the optional arguments `abs_tol` and `rel_tol`).

x — real(kind=*wp*), intent(out)

Output: the point at which this procedure terminated. If no errors are reported, `x` contains an estimate of x_{min}.

f — real(kind=*wp*), intent(out)

Output: the value of the function f at the point returned in `x`; i.e., the estimate of the minimum, if no errors are reported.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

use_deriv — logical, intent(in), optional

Input: specifies whether or not the first derivative is provided in the user-supplied procedure `fun`.

If `use_deriv` = `.true.` (the default), then the first derivative is assumed to be provided in `fun` via its optional argument `g`;

if `use_deriv` = `.false.`, then it is assumed that the first derivative is not provided.

Default: `use_deriv` = `.true.`.

rel_tol — real(kind=*wp*), intent(in), optional

Input: ε_r, the relative tolerance to which x_{min} is required (see Section 6.1).

Constraints: `EPSILON(1.0_`*wp*`)` ≤ `rel_tol` < 1.0.

Default: `rel_tol` = `SQRT(EPSILON(1.0_`*wp*`))`.

abs_tol — real(kind=*wp*), intent(in), optional

Input: ε_{abs}, the absolute tolerance to which x_{min} is required (see Section 6.1).

Constraints: `abs_tol` ≥ 0.0.

Default: `abs_tol` = `SQRT(EPSILON(1.0_`*wp*`))`.

max_fun_eval — integer, intent(in), optional

Input: the maximum number of calls to be made to `fun`.

Constraints: `max_fun_eval` ≥ 3.

Default: `max_fun_eval` = 30.

num_fun_eval — integer, intent(out), optional

Output: the actual number of calls made to `fun`.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

Failures (error%level = 2):

error%code	Description
201	The limiting number of calls to `fun` was reached. A solution that satisfies the accuracies `rel_tol` and `abs_tol` was not found within `max_fun_eval` calls to `fun`. Check whether `fun` has been coded correctly. If it is correct, restart `nag_uv_min_sol` (preferably with the values of `a` and `b` returned from the previous call). We do not recommend putting the call statement within a loop which resets `max_fun_eval`.

Warnings (error%level = 1):

error%code	Description
101	Possible error in `fun`. A solution has been found but it appears that the code supplied for the first derivative in `fun` is incorrect. Check whether the code for the optional argument `g` in `fun` is correct. You could also try to find a solution without using the first derivative (by calling `nag_uv_min_sol` with the optional argument `use_deriv` set to `.false.`).

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Accuracy

If $f(x)$ is δ-unimodal (see [1]) for some $\delta < Tol(x)$, where $Tol(x) = \varepsilon_r|x| + \varepsilon_{abs}$, then on exit x approximates x_{min}, the location of the minimum of $f(x)$, in the original interval $[a, b]$ with an error less than $3 \times Tol(x)$.

6.2 Timing

Timing depends on the behaviour of $f(x)$, the accuracy demanded and the length of the interval $[a, b]$. Unless $f(x)$ and (if appropriate) df/dx can be evaluated very quickly, the run time will usually be dominated by the time spent in `fun`.

6.3 Termination Criteria

If $f(x)$ has more than one minimum in the original interval $[a, b]$, an approximation x (and improved bounds a and b) will be determined for the location of one of the minima.

If an x is found such that $f(x-\delta_1) > f(x) < f(x+\delta_2)$ for some $\delta_1, \delta_2 \geq Tol(x)$, the interval $[x-\delta_1, x+\delta_2]$ will be regarded as containing x_{min}, even if $f(x)$ is less than $f(x-\delta_1)$ and $f(x+\delta_2)$ only due to rounding errors in `fun`. Therefore `fun` should be programmed to calculate $f(x)$ and (if appropriate) df/dx as accurately as possible near x_{min}, i.e., where df/dx is near zero, thereby reducing the risk of termination at a spurious minimum.

Example 1: Univariate Minimization With and Without Derivatives

In this example we use `nag_uv_min_sol` to find the location of the minimum of the function $f(x) = \sin(x)/x$ in the interval $[3.5, 5.0]$ using the first derivative.

We then find the location of the minimum – starting from the same original interval – without using the first derivative. This is done to show that the answers are the same in both cases (as expected). However, the number of calls to the user-supplied function `fun` increases when the first derivative is not used.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
MODULE uv_min_ex01_mod
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Default Accessibility ..
  PUBLIC
  ! .. Intrinsic Functions ..
  INTRINSIC KIND, PRESENT
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)

CONTAINS

  SUBROUTINE fun(x,f,g)
    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Intrinsic Functions ..
    INTRINSIC COS, SIN
    ! .. Scalar Arguments ..
    REAL (wp), INTENT (OUT) :: f
    REAL (wp), OPTIONAL, INTENT (OUT) :: g
    REAL (wp), INTENT (IN) :: x
    ! .. Executable Statements ..

    f = SIN(x)/x
    IF (PRESENT(g)) g = (COS(x)-f)/x
  END SUBROUTINE fun
END MODULE uv_min_ex01_mod

PROGRAM nag_uv_min_ex01

  ! Example Program Text for nag_uv_min
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_uv_min, ONLY : nag_uv_min_sol
  USE uv_min_ex01_mod, ONLY : fun, wp
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Local Scalars ..
  INTEGER :: num_fun_eval
  REAL (wp) :: a, b, f, fa, fb, x
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_uv_min_ex01'

  ! Find the minimum of fun in the interval [a,b], using derivatives
```

```
    a = 3.5_wp
    b = 5.0_wp

    CALL fun(a,fa)
    CALL fun(b,fb)
    WRITE (nag_std_out,'(/,A,''['',F8.5,'','',F8.5,'']'')') &
     ' Original interval is ', a, b
    WRITE (nag_std_out,'(''  f('',F4.2,'') = '',1PE12.5)') a, fa
    WRITE (nag_std_out,'(''  f('',F4.2,'') = '',1PE12.5)') b, fb

    CALL nag_uv_min_sol(fun,a,b,x,f,num_fun_eval=num_fun_eval)

    WRITE (nag_std_out,'(/,A)') ' Minimization with derivatives:'
    WRITE (nag_std_out,'(A,1PE12.5)') &
     '  Estimated minimum located at x  . . = ', x
    WRITE (nag_std_out,'(A,1PE12.5)') &
     '  The function value at the above x . = ', f
    WRITE (nag_std_out,'(A,1PE12.5)') &
     '  Length of the final interval  (b-a) = ', b - a
    WRITE (nag_std_out,'(A,I3)') '  No of function calls  . . . . . . . = ', &
     num_fun_eval

    ! Find the minimum of fun in the interval [a,b], without derivatives

    a = 3.5_wp
    b = 5.0_wp

    CALL nag_uv_min_sol(fun,a,b,x,f,use_deriv=.FALSE., &
     num_fun_eval=num_fun_eval)

    WRITE (nag_std_out,'(/,A)') ' Minimization without derivatives:'
    WRITE (nag_std_out,'(A,1PE12.5)') &
     '  Estimated minimum located at x  . . = ', x
    WRITE (nag_std_out,'(A,1PE12.5)') &
     '  The function value at the above x . = ', f
    WRITE (nag_std_out,'(A,1PE12.5)') &
     '  Length of the final interval  (b-a) = ', b - a
    WRITE (nag_std_out,'(A,I3)') '  No of function calls  . . . . . . . = ', &
     num_fun_eval
  END PROGRAM nag_uv_min_ex01
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_uv_min_ex01

 Original interval is [ 3.50000, 5.00000]
  f(3.50) = -1.00224E-01
  f(5.00) = -1.91785E-01

 Minimization with derivatives:
  Estimated minimum located at x  . . =  4.49341E+00
  The function value at the above x . = -2.17234E-01
  Length of the final interval  (b-a) =  8.18582E-08
  No of function calls  . . . . . . . =   6

 Minimization without derivatives:
  Estimated minimum located at x  . . =  4.49341E+00
  The function value at the above x . = -2.17234E-01
  Length of the final interval  (b-a) =  1.63716E-07
  No of function calls  . . . . . . . =  10
```

Additional Examples

Not all example programs supplied with NAG *fl*90 appear in full in this module document. The following additional examples, associated with this module, are available.

`nag_uv_min_ex02`

Univariate minimization with derivatives.

`nag_uv_min_ex03`

Univariate minimization without derivatives.

References

[1] Gill P E, Murray W and Wright M H (1981) *Practical Optimization* Academic Press