NAG Library Function Document

nag_interval_zero_cont_func (c05avc)

1 Purpose

nag_interval_zero_cont_func (c05avc) attempts to locate an interval containing a simple zero of a continuous function using a binary search. It uses reverse communication for evaluating the function.

2 Specification

```
#include <nag.h>
#include <nagc05.h>
void nag_interval_zero_cont_func (double *x, double fx, double *h,
     double boundl, double boundu, double *y, double c[], Integer *ind,
     NagError *fail)
```

3 Description

You must supply an initial point **x** and a step **h**. nag_interval_zero_cont_func (c05avc) attempts to locate a short interval $[\mathbf{x}, \mathbf{y}] \subset [\mathbf{boundl}, \mathbf{boundu}]$ containing a simple zero of $f(x)$.

(On exit we may have $\mathbf{x} > \mathbf{y}$; **x** is determined as the first point encountered in a binary search where the sign of $f(x)$ differs from the sign of $f(x)$ at the initial input point **x**.) The function attempts to locate a zero of $f(x)$ using **h**, $0.1 \times \mathbf{h}$, $0.01 \times \mathbf{h}$ and $0.001 \times \mathbf{h}$ in turn as its basic step before quitting with an error exit if unsuccessful.

nag_interval_zero_cont_func (c05avc) returns to the calling program for each evaluation of $f(x)$. On each return you should set $\mathbf{fx} = f(\mathbf{x})$ and call nag_interval_zero_cont_func (c05avc) again.

4 References

None.

5 Arguments

Note: this function uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the argument **ind**. Between intermediate exits and re-entries, **all arguments other than fx must remain unchanged**.

1: **x** – double * *Input/Output*

On initial entry: the best available approximation to the zero.

Constraint: **x** must lie in the closed interval $[\mathbf{boundl}, \mathbf{boundu}]$ (see below).

On intermediate exit: contains the point at which f must be evaluated before re-entry to the function.

On final exit: contains one end of an interval containing the zero, the other end being in **y**, unless an error has occurred. If **fail.code** = NE_ZERO_NOT_FOUND, **x** and **y** are the end points of the largest interval searched. If a zero is located exactly, its value is returned in **x** (and in **y**).

2: **fx** – double *Input*

On initial entry: if $\mathbf{ind} = 1$, **fx** need not be set.

If $\mathbf{ind} = -1$, **fx** must contain $f(\mathbf{x})$ for the initial value of **x**.

On intermediate re-entry: must contain $f(\mathbf{x})$ for the current value of **x**.

3: **h** – double * *Input/Output*

On initial entry: a basic step size which is used in the binary search for an interval containing a zero. The basic step sizes $\mathbf{h}, 0.1 \times \mathbf{h}$, $0.01 \times \mathbf{h}$ and $0.001 \times \mathbf{h}$ are used in turn when searching for the zero.

Constraint: either $\mathbf{x} + \mathbf{h}$ or $\mathbf{x} - \mathbf{h}$ must lie inside the closed interval $[\mathbf{boundl}, \mathbf{boundu}]$.

h must be sufficiently large that $\mathbf{x} + \mathbf{h} \neq \mathbf{x}$ on the computer.

On final exit: is undefined.

4: **boundl** – double *Input*

5: **boundu** – double *Input*

On initial entry: **boundl** and **boundu** must contain respectively lower and upper bounds for the interval of search for the zero.

Constraint: **boundl** $<$ **boundu**.

6: **y** – double * *Input/Output*

On initial entry: need not be set.

On final exit: contains the closest point found to the final value of **x**, such that $f(\mathbf{x}) \times f(\mathbf{y}) \leq 0.0$. If a value **x** is found such that $f(\mathbf{x}) = 0$, then $\mathbf{y} = \mathbf{x}$. On final exit with **fail.code** = NE_ZERO_NOT_FOUND, **x** and **y** are the end points of the largest interval searched.

7: **c**[**11**] – double *Communication Array*

On initial entry: need not be set.

On final exit: if **fail.code** = NE_NOERROR or NE_ZERO_NOT_FOUND, **c**[0] contains $f(\mathbf{y})$.

8: **ind** – Integer * *Input/Output*

On initial entry: must be set to 1 or -1.

ind $= 1$
: **fx** need not be set.

ind $= -1$
: **fx** must contain $f(\mathbf{x})$.

On intermediate exit: contains 2 or 3. The calling program must evaluate f at **x**, storing the result in **fx**, and re-enter nag_interval_zero_cont_func (c05avc) with all other arguments unchanged.

On final exit: contains 0.

Constraint: on entry **ind** $= -1$, 1, 2 or 3.

9: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **ind** = ⟨*value*⟩.
Constraint: **ind** = −1, 1, 2 or 3.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

NE_REAL_2

On entry, **boundl** = ⟨*value*⟩ and **boundu** = ⟨*value*⟩.
Constraint: **boundl** < **boundu**.

On entry, **h** is too small for use as a perturbation of **x**: **x** = ⟨*value*⟩ and **h** = ⟨*value*⟩.

NE_REAL_3

On entry, **x** = ⟨*value*⟩, **boundl** = ⟨*value*⟩ and **boundu** = ⟨*value*⟩.
Constraint: **boundl** ≤ **x** ≤ **boundu**.

NE_REAL_4

On entry, **x** + **h** and **x** − **h** both lie outside the interval [**boundl**, **boundu**]: **x** = ⟨*value*⟩, **h** = ⟨*value*⟩, **boundl** = ⟨*value*⟩ and **boundu** = ⟨*value*⟩.

NE_ZERO_NOT_FOUND

An interval containing the zero could not be found.

7 Accuracy

nag_interval_zero_cont_func (c05avc) is not intended to be used to obtain accurate approximations to the zero of $f(x)$ but rather to locate an interval containing a zero. This interval can then be used as input to an accurate rootfinder such as nag_zero_cont_func_brent (c05ayc) or nag_zero_cont_func_brent_rcomm (c05azc). The size of the interval determined depends somewhat unpredictably on the choice of **x** and **h**. The closer **x** is to the root and the **smaller** the initial value of **h**, then, in general, the smaller (more accurate) the interval determined; however, the accuracy of this statement depends to some extent on the behaviour of $f(x)$ near $x =$ **x** and on the size of **h**.

8 Parallelism and Performance

nag_interval_zero_cont_func (c05avc) is not threaded in any implementation.

9 Further Comments

For most problems, the time taken on each call to nag_interval_zero_cont_func (c05avc) will be negligible compared with the time spent evaluating $f(x)$ between calls to nag_interval_zero_cont_func (c05avc). However, the initial value of **x** and **h** will clearly affect the timing. The closer **x** is to the root, and the **larger** the initial value of **h** then the less time taken. (However taking a large **h** can affect the accuracy and reliability of the function, see below.)

You are expected to choose **boundl** and **boundu** as physically (or mathematically) realistic limits on the interval of search. For example, it may be known, from physical arguments, that no zero of $f(x)$ of

interest will lie outside [**boundl**, **boundu**]. Alternatively, $f(x)$ may be more expensive to evaluate for some values of **x** than for others and such expensive evaluations can sometimes be avoided by careful choice of **boundl** and **boundu**.

The choice of **boundl** and **boundu** affects the search only in that these values provide physical limitations on the search values and that the search is terminated if it seems, from the available information about $f(x)$, that the zero lies outside [**boundl**, **boundu**]. In this case (**fail.code** = NE_ZERO_NOT_FOUND on exit), only one of f(**boundl**) and f(**boundu**) may have been evaluated and a zero close to the other end of the interval could be missed. The actual interval searched is returned in the arguments **x** and **y** and you can call nag_interval_zero_cont_func (c05avc) again to search the remainder of the original interval.

Though nag_interval_zero_cont_func (c05avc) is intended primarily for determining an interval containing a zero of $f(x)$, it may be used to shorten a known interval. This could be useful if, for example, a large interval containing the zero is known and it is also known that the root lies close to one end of the interval; by setting **x** to this end of the interval and **h** small, a short interval will usually be determined. However, it is worth noting that once any interval containing a zero has been determined, a call to nag_zero_cont_func_brent_rcomm (c05azc) will usually be the most efficient way to calculate an interval of specified length containing the zero. To assist in this determination, the information in **fx** and in **x**, **y** and **c**[0] on successful exit from nag_interval_zero_cont_func (c05avc) is in the correct form for a call to function nag_zero_cont_func_brent_rcomm (c05azc) with **ind** $= -1$.

If the calculation terminates because $f(\mathbf{x}) = 0.0$, then on return **y** is set to **x**. (In fact, $\mathbf{y} = \mathbf{x}$ on return only in this case.) In this case, there is no guarantee that the value in **x** corresponds to a **simple** zero and you should check whether it does.

One way to check this is to compute the derivative of f at the point **x**, preferably analytically, or, if this is not possible, numerically, perhaps by using a central difference estimate. If $f'(\mathbf{x}) = 0.0$, then **x** must correspond to a multiple zero of f rather than a simple zero.

10 Example

This example finds a sub-interval of $[0.0, 4.0]$ containing a simple zero of $x^2 - 3x + 2$. The zero nearest to 3.0 is required and so we set $\mathbf{x} = 3.0$ initially.

10.1 Program Text

```
/* nag_interval_zero_cont_func (c05avc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <math.h>
#include <nagc05.h>

int main(void)
{
  /* Scalars */
  Integer exit_status = 0;
  double boundl, boundu, fx, h, x, y;
  Integer ind;
  /* Arrays */
  double c[11];
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_interval_zero_cont_func (c05avc) Example Program Results\n");
```

```
  x = 3.0;
  h = 0.1;
  boundl = 0.0;
  boundu = 4.0;
  ind = 1;
  fx = 0.0;
  /* nag_interval_zero_cont_func (c05avc).
   * Locates an interval containing a simple zero of a continuous
   * function using binary search and reverse communication.
   */
  while (ind != 0) {
    nag_interval_zero_cont_func(&x, fx, &h, boundl, boundu, &y, c, &ind,
                                &fail);

    if (ind != 0)
      fx = pow(x, 2) - 3.0 * x + 2.0;
  }

  if (fail.code == NE_NOERROR) {
    printf("Interval containing root is [x,y], where\n");
    printf("x = %12.4f, y = %12.4f\n", x, y);
    printf("Values of f at x and y are\n");
    printf("f(x) = %12.2f, f(y) = %12.2f\n", fx, c[0]);
  }
  else {
    printf("%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

END:
  return exit_status;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_interval_zero_cont_func (c05avc) Example Program Results
Interval containing root is [x,y], where
x =       1.7000, y =       2.5000
Values of f at x and y are
f(x) =        -0.21, f(y) =         0.75
```