nag_zero_cont_func_bd_1 (c05sdc)

1. Purpose

nag_zero_cont_func_bd_1 (c05sdc) locates a zero of a continuous function in a given interval by a combination of the methods of linear interpolation, extrapolation and bisection.

2. Specification

```
#include <nag.h>
#include <nagc05.h>

void nag_zero_cont_func_bd_1(double a, double b, double *x,
                     double (*f)(double x, Nag_User *comm), double xtol,
                     double ftol, Nag_User *comm, NagError *fail)
```

3. Description

The routine attempts to obtain an approximation to a simple zero of the function $f(x)$ given an initial interval $[a,b]$ such that $f(a)\times f(b)\leq 0$. The zero is found by a modified version of procedure 'zeroin' given by Bus and Dekker (1975). The approximation x to the zero α is determined so that one or both of the following criteria are satisfied:

(i) $|x-\alpha| <$ **xtol**,
(ii) $|f(x)| <$ **ftol**.

The routine combines the methods of bisection, linear interpolation and linear extrapolation (see Dahlquist and Bjorck (1974)), to find a sequence of sub-intervals of the initial interval such that the final interval $[x,\, y]$ contains the zero and is small enough to satisfy the tolerance specified by **xtol**. Note that, since the intervals $[x,\, y]$ are determined only so that they contain a change of sign of f, it is possible that the final interval may contain a discontinuity or a pole of f (violating the requirement that f be continuous). If the sign change is likely to correspond to a pole of f then the routine gives an error return.

4. Parameters

a

Input: the lower bound of the interval, a.

b

Input: the upper bound of the interval, b.
Constraint: $\mathbf{b} \neq \mathbf{a}$.

x

Output: the approximation to the zero.

f

The function **f**, supplied by the user, must evaluate the function f whose zero is to be determined.
The specification of **f** is:

```
double f(double x, Nag_User *comm)
```

x

Input: the point x at which the function must be evaluated.

comm

Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer
Input/Output: the pointer **comm->p** should be cast to the required type, e.g. `struct user *s = (struct user *)comm->p`, to obtain the original object's address with appropriate type. (See the argument **comm** below.)

xtol
Input: the absolute tolerance to which the zero is required (see Section 3).
Constraint: **xtol** > 0.0.

ftol
Input: a value such that if $|f(x)| <$ **ftol**, x is accepted as the zero. **ftol** may be specified as 0.0 (see Section 6).

comm
Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer
Input/Output: the pointer **p**, of type Pointer, allows the user to communicate information to and from the user-defined function **f()**. An object of the required type should be declared by the user, e.g. a structure, and its address assigned to the pointer **p** by means of a cast to Pointer in the calling program, e.g. `comm.p = (Pointer)&s`. The type pointer will be `void *` with a C compiler that defines `void *` and `char *` otherwise.

fail
The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

NE_2_REAL_ARG_EQ
On entry, **a** = ⟨*value*⟩ while **b** = ⟨*value*⟩. These parameters must satisfy **a** $\neq$ **b**.

NE_REAL_ARG_LE
On entry, **xtol** must not be less than or equal to 0.0: **xtol** = ⟨*value*⟩.

NE_FUNC_END_VAL
On entry, **f**(⟨*value*⟩) and **f**(⟨*value*⟩) have the same sign, with **f**(⟨*value*⟩) $\neq 0.0$.

NE_PROBABLE_POLE
Indicates that the function values in the interval [**a**,**b**] might contain a pole rather than a zero. Reducing **xtol** may help in distinguishing between a pole and a zero.

NE_XTOL_TOO_SMALL
No further improvement in the solution is possible. **xtol** is too small: **xtol** = ⟨*value*⟩.

6. Further Comments

The time taken by the routine depends primarily on the time spent evaluating **f** (see Section 4).

6.1. Accuracy

This depends on the value of **xtol** and **ftol**. If full machine accuracy is required, they may be set very small, resulting in an error exit with error exit of **NE_XTOL_TOO_SMALL**, although this may involve many more iterations than a lesser accuracy. The user is recommended to set **ftol** $= 0.0$ and to use **xtol** to control the accuracy, unless there is prior knowledge of the size of $f(x)$ for values of x near the zero.

6.2. References

Bus J C P and Dekker T J (1975) Two Efficient Algorithms with Guaranteed Convergence for Finding a Zero of a Function *ACM Trans. Math. Softw.* **1** 330–345.
Dahlquist G and Bjorck A (1974) *Numerical Methods* Prentice-Hall.

7. See Also

None.

8. Example

The example program below calculates the zero of $e^{-x} - x$ within the interval $[0, 1]$ to approximately 5 decimal places.

8.1. Program Text

```
/* nag_zero_cont_func_bd_1(c05sdc) Example Program
 *
 * Copyright 1998 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 */

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

#ifdef NAG_PROTO
static double f(double x, Nag_User *comm);
#else
static double f();
#endif

main()
{
  double a, b;
  double x, ftol, xtol;
  static NagError fail;
  Nag_User comm;

  Vprintf("c05sdc Example Program Results\n");
  a = 0.0;
  b = 1.0;
  xtol = 1e-05;
  ftol = 0.0;
  c05sdc(a, b, &x, f, xtol, ftol, &comm, &fail);
  if (fail.code == NE_NOERROR)
    {
      Vprintf("Zero = %12.5f\n",x);
      exit(EXIT_SUCCESS);
    }
  else
    {
      Vprintf("%s\n", fail.message);
      if (fail.code == NE_XTOL_TOO_SMALL ||
          fail.code == NE_PROBABLE_POLE)
        Vprintf("Final point = %12.5f\n",x);
      exit(EXIT_FAILURE);
    }
}

#ifdef NAG_PROTO
static double f(double x, Nag_User *comm)
#else
     static double f(x, comm)
     double x;
     Nag_User *comm;
#endif
{
  return exp(-x)-x;
}
```

8.2. Program Data

None.

8.3. Program Results

```
c05sdc Example Program Results
Zero =      0.56714
```