NAG Library Function Document

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

nag_zero_cont_func_bd_1 (c05sdc) 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 function combines the methods of bisection, linear interpolation and linear extrapolation (see Dahlquist and Björck (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 function gives an error return.

4 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. Software* **1** 330–345

Dahlquist G and Björck Å (1974) *Numerical Methods* Prentice–Hall

5 Arguments

1: **a** – double *Input*

On entry: the lower bound of the interval, a.

2: **b** – double *Input*

On entry: the upper bound of the interval, b.

Constraint: **b** $\neq$ **a**.

3: **x** – double * *Output*

On exit: the approximation to the zero.

4: **f** – function, supplied by the user *External Function*

f must evaluate the function f whose zero is to be determined.

The specification of **f** is:

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

1: **x** – double *Input*

On entry: the point x at which the function must be evaluated.

2: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: 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.)

5: **xtol** – double *Input*

On entry: the absolute tolerance to which the zero is required (see Section 3).

Constraint: **xtol** > 0.0.

6: **ftol** – double *Input*

On entry: a value such that if $|f(x)| <$ **ftol**, x is accepted as the zero. **ftol** may be specified as 0.0 (see Section 9).

7: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: the pointer **comm→p**, of type Pointer, allows you to communicate information to and from **f**(). You must declare an object of the required type, e.g., a structure, and its address assigned to the pointer **comm→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.

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_2_REAL_ARG_EQ

On entry, **a** = ⟨*value*⟩ while **b** = ⟨*value*⟩. These arguments must satisfy **a** ≠ **b**.

NE_FUNC_END_VAL

On entry, **f**(⟨*value*⟩) and **f**(⟨*value*⟩) have the same sign, with **f**(⟨*value*⟩) ≠ 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_REAL_ARG_LE

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

NE_XTOL_TOO_SMALL

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

7 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. You are recommended to set **ftol** = 0.0 and to use **xtol** to control the accuracy, unless you have considerable knowledge of the size of $f(x)$ for values of x near the zero.

8 Parallelism and Performance

Not applicable.

9 Further Comments

The time taken by nag_zero_cont_func_bd_1 (c05sdc) depends primarily on the time spent evaluating **f** (see Section 5).

10 Example

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

10.1 Program Text

```
/* nag_zero_cont_func_bd_1 (c05sdc) Example Program.
 *
 * Copyright 1998 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 * Mark 7 revised, 2001.
 */

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

#ifdef __cplusplus
extern "C" {
#endif
static double NAG_CALL f(double x, Nag_User *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{
  Integer  exit_status = 0;
  double   a, b;
  double   x, ftol, xtol;
  NagError fail;
  Nag_User comm;

  INIT_FAIL(fail);
```

```
  printf("nag_zero_cont_func_bd_1 (c05sdc) Example Program Results\n");

  a = 0.0;
  b = 1.0;
  xtol = 1e-05;
  ftol = 0.0;

  /* nag_zero_cont_func_bd_1 (c05sdc).
   * Zero of a continuous function of one variable,
   * thread-safe
   */
  nag_zero_cont_func_bd_1(a, b, &x, f, xtol, ftol, &comm, &fail);
  if (fail.code == NE_NOERROR)
    {
      printf("Zero = %12.5f\n", x);
    }
  else
    {
      printf("%s\n", fail.message);
      if (fail.code == NE_XTOL_TOO_SMALL ||
          fail.code == NE_PROBABLE_POLE)
        printf("Final point = %12.5f\n", x);
      exit_status = 1;
      goto END;
    }

 END:
  return exit_status;
}

static double NAG_CALL f(double x, Nag_User *comm)
{
  return exp(-x)-x;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_zero_cont_func_bd_1 (c05sdc) Example Program Results
Zero =      0.56714
```