NAG Library Function Document

nag_zero_cont_func_brent_bsrch (c05agc)

1 Purpose

nag_zero_cont_func_brent_bsrch (c05agc) locates a simple zero of a continuous function from a given starting value, using a binary search to locate an interval containing a zero of the function, then a combination of the methods of nonlinear interpolation, linear extrapolation and bisection to locate the zero precisely.

2 Specification

```
#include <nag.h>
#include <nagc05.h>
void nag_zero_cont_func_brent_bsrch (double *x, double h, double xtol,
     double ftol,
     double  (*f)(double xx, Nag_Comm *comm),
     double *a, double *b, Nag_Comm *comm, NagError *fail)
```

3 Description

nag_zero_cont_func_brent_bsrch (c05agc) attempts to locate an interval $[a, b]$ containing a simple zero of the function $f(x)$ by a binary search starting from the initial point $x = \mathbf{x}$ and using repeated calls to nag_interval_zero_cont_func (c05avc). If this search succeeds, then the zero is determined to a user-specified accuracy by a call to nag_zero_cont_func_brent (c05ayc). The specifications of functions nag_interval_zero_cont_func (c05avc) and nag_zero_cont_func_brent (c05ayc) should be consulted for details of the methods used.

The approximation x to the zero α is determined so that at least one of the following criteria is satisfied:

(i) $|x - \alpha| \leq$ **xtol**,

(ii) $|f(x)| \leq$ **ftol**.

4 References

Brent R P (1973) *Algorithms for Minimization Without Derivatives* Prentice–Hall

5 Arguments

1: **x** – double * *Input/Output*

On entry: an initial approximation to the zero.

On exit: if **fail.code** = NE_NOERROR or NW_TOO_MUCH_ACC_REQUESTED, **x** is the final approximation to the zero.

If **fail.code** = NE_PROBABLE_POLE, **x** is likely to be a pole of $f(x)$.

Otherwise, **x** contains no useful information.

2: **h** – double *Input*

On entry: a step length for use in the binary search for an interval containing the zero. The maximum interval searched is $[\mathbf{x} - 256.0 \times \mathbf{h}, \mathbf{x} + 256.0 \times \mathbf{h}]$.

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

3: **xtol** – double *Input*

On entry: the termination tolerance on x (see Section 3).

Constraint: **xtol** > 0.0.

4: **ftol** – double *Input*

On entry: a value such that if $|f(x)| \leq$ **ftol**, x is accepted as the zero. **ftol** may be specified as 0.0 (see Section 7).

5: **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 xx, Nag_Comm *comm)
```

1: **xx** – double *Input*

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

2: **comm** – Nag_Comm * *Communication Structure*

Pointer to structure of type Nag_Comm; the following members are relevant to **f**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_zero_cont_func_brent_bsrch (c05agc) you may allocate memory and initialize these pointers with various quantities for use by **f** when called from nag_zero_cont_func_brent_bsrch (c05agc) (see Section 3.2.1.1 in the Essential Introduction).

6: **a** – double * *Output*
7: **b** – double * *Output*

On exit: the lower and upper bounds respectively of the interval resulting from the binary search. If the zero is determined exactly such that $f(x) = 0.0$ or is determined so that $|f(x)| \leq$ **ftol** at any stage in the calculation, then on exit **a** $=$ **b** $= x$.

8: **comm** – Nag_Comm * *Communication Structure*

The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

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

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

6 Error Indicators and Warnings

NE_BAD_PARAM

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

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.

NE_PROBABLE_POLE

Solution may be a pole rather than a zero.

NE_REAL

On entry, **xtol** = ⟨*value*⟩.
Constraint: **xtol** > 0.0.

NE_REAL_2

On entry, **x** = ⟨*value*⟩ and **h** = ⟨*value*⟩.
Constraint: **x** + **h** ≠ **x** (to machine accuracy).

NE_ZERO_NOT_FOUND

An interval containing the zero could not be found.

NW_TOO_MUCH_ACC_REQUESTED

The tolerance **xtol** has been set too small for the problem being solved. However, the value **x** returned is a good approximation to the zero. **xtol** = ⟨*value*⟩.

7 Accuracy

The levels of accuracy depend on the values of **xtol** and **ftol**. If full machine accuracy is required, they may be set very small, resulting in an exit with **fail.code** = NW_TOO_MUCH_ACC_REQUESTED, 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_brent_bsrch (c05agc) depends primarily on the time spent evaluating **f** (see Section 5). The accuracy of the initial approximation **x** and the value of **h** will have a somewhat unpredictable effect on the timing.

If it is important to determine an interval of relative length less than 2 × **xtol** containing the zero, or if **f** is expensive to evaluate and the number of calls to **f** is to be restricted, then use of nag_interval_zero_cont_func (c05avc) followed by nag_zero_cont_func_brent_rcomm (c05azc) is recommended. Use of this combination is also recommended when the structure of the problem to be solved does not permit a simple **f** to be written: the reverse communication facilities of these functions are more flexible than the direct communication of **f** required by nag_zero_cont_func_brent_bsrch (c05agc).

If the iteration terminates with successful exit and **a** = **b** = **x** there is no guarantee that the value returned 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 calculates an approximation to the zero of $x - e^{-x}$ using a tolerance of **xtol** = 1.0e−5 starting from **x** = 1.0 and using an initial search step **h** = 0.1.

10.1 Program Text

```
/* nag_zero_cont_func_brent_bsrch (c05agc) Example Program.
 *
 * Copyright 2006 Numerical Algorithms Group.
 *
 * Mark 9, 2009.
 */

#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_Comm *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{
  /* Scalars */
  Integer  exit_status = 0;
  double   a, b, eps, eta, h, x;
  NagError fail;
  Nag_Comm user;

  INIT_FAIL(fail);

  printf("nag_zero_cont_func_brent_bsrch (c05agc) Example Program Results\n");

  x = 1.0;
  h = 0.1;
  eps = 1e-05;
  eta = 0.0;
  /* nag_zero_cont_func_brent_bsrch (c05agc).
   * Locates a simple zero of a continuous function of one variable,
   * binary search for an interval containing a zero.
   */
  nag_zero_cont_func_brent_bsrch(&x, h, eps, eta, f, &a, &b, &user, &fail);
  if (fail.code == NE_NOERROR)
    {
      printf("Root is %13.5f\n", x);
      printf("Interval searched is [%8.5f,%8.5f]\n", a, b);
    }
  else
    {
      printf("%s\n", fail.message);
      if (fail.code == NE_PROBABLE_POLE ||
          fail.code == NW_TOO_MUCH_ACC_REQUESTED)
        printf("Final value = %13.5f\n", x);
      exit_status = 1;
      goto END;
    }

 END:
  return exit_status;
}

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

10.2 Program Data

None.

10.3 Program Results

```
nag_zero_cont_func_brent_bsrch (c05agc) Example Program Results
Root is       0.56714
Interval searched is [ 0.50000, 0.90000]
```