

## NAG Library Function Document

### nag\_zero\_cont\_func\_brent\_rcomm (c05azc)

#### 1 Purpose

nag\_zero\_cont\_func\_brent\_rcomm (c05azc) locates a simple zero of a continuous function in a given interval by using Brent's method, which is a combination of nonlinear interpolation, linear extrapolation and bisection. It uses reverse communication for evaluating the function.

#### 2 Specification

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

void nag_zero_cont_func_brent_rcomm (double *x, double *y, double fx,
    double tolx, Nag_ErrorControl ir, double c[], Integer *ind,
    NagError *fail)
```

#### 3 Description

You must supply  $\mathbf{x}$  and  $\mathbf{y}$  to define an initial interval  $[a, b]$  containing a simple zero of the function  $f(x)$  (the choice of  $\mathbf{x}$  and  $\mathbf{y}$  must be such that  $f(\mathbf{x}) \times f(\mathbf{y}) \leq 0.0$ ). The function combines the methods of bisection, nonlinear 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  $[\mathbf{x}, \mathbf{y}]$  contains the zero and  $|\mathbf{x} - \mathbf{y}|$  is less than some tolerance specified by **tolx** and **ir** (see Section 5). In fact, since the intermediate intervals  $[\mathbf{x}, \mathbf{y}]$  are determined only so that  $f(\mathbf{x}) \times f(\mathbf{y}) \leq 0.0$ , it is possible that the final interval may contain a discontinuity or a pole of  $f$  (violating the requirement that  $f$  be continuous). nag\_zero\_cont\_func\_brent\_rcomm (c05azc) checks if the sign change is likely to correspond to a pole of  $f$  and gives an error return in this case.

A feature of the algorithm used by this function is that unlike some other methods it guarantees convergence within about  $(\log_2[(b - a)/\delta])^2$  function evaluations, where  $\delta$  is related to the argument **tolx**. See Brent (1973) for more details.

nag\_zero\_cont\_func\_brent\_rcomm (c05azc) 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\_zero\_cont\_func\_brent\_rcomm (c05azc) again.

The function is a modified version of procedure 'zeroin' given by Brent (1973).

#### 4 References

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

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

**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  $\mathbf{fx}$  must remain unchanged**.

- 1: **x** – double \* *Input/Output*  
 2: **y** – double \* *Input/Output*

*On initial entry:* **x** and **y** must define an initial interval  $[a, b]$  containing the zero, such that  $f(\mathbf{x}) \times f(\mathbf{y}) \leq 0.0$ . It is not necessary that  $\mathbf{x} < \mathbf{y}$ .

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

*On final exit:* **x** and **y** define a smaller interval containing the zero, such that  $f(\mathbf{x}) \times f(\mathbf{y}) \leq 0.0$ , and  $|\mathbf{x} - \mathbf{y}|$  satisfies the accuracy specified by **tolx** and **ir**, unless an error has occurred. If **fail.code** = NE\_PROBABLE\_POLE, **x** and **y** generally contain very good approximations to a pole; if **fail.code** = NW\_TOO\_MUCH\_ACC\_REQUESTED, **x** and **y** generally contain very good approximations to the zero (see Section 6). If a point **x** is found such that  $f(\mathbf{x}) = 0.0$ , then on final exit  $\mathbf{x} = \mathbf{y}$  (in this case there is no guarantee that **x** is a simple zero). In all cases, the value returned in **x** is the better approximation to the zero.

- 3: **fx** – double *Input*

*On initial entry:* if **ind** = 1, **fx** need not be set.

If **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**.

- 4: **tolx** – double *Input*

*On initial entry:* the accuracy to which the zero is required. The type of error test is specified by **ir**.

*Constraint:* **tolx** > 0.0.

- 5: **ir** – Nag\_ErrorControl *Input*

*On initial entry:* indicates the type of error test.

**ir** = Nag\_Mixed

The test is:  $|\mathbf{x} - \mathbf{y}| \leq 2.0 \times \mathbf{tolx} \times \max(1.0, |\mathbf{x}|)$ .

**ir** = Nag\_Absolute

The test is:  $|\mathbf{x} - \mathbf{y}| \leq 2.0 \times \mathbf{tolx}$ .

**ir** = Nag\_Relative

The test is:  $|\mathbf{x} - \mathbf{y}| \leq 2.0 \times \mathbf{tolx} \times |\mathbf{x}|$ .

*Suggested value:* **ir** = Nag\_Mixed.

*Constraint:* **ir** = Nag\_Mixed, Nag\_Absolute or Nag\_Relative.

- 6: **c[17]** – double *Input/Output*

*On initial entry:* if **ind** = 1, no elements of **c** need be set.

If **ind** = -1, **c[0]** must contain  $f(\mathbf{y})$ , other elements of **c** need not be set.

*On final exit:* is undefined.

- 7: **ind** – Integer \* *Input/Output*

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

**ind** = 1

**fx** and **c[0]** need not be set.

**ind** = -1

**fx** and **c[0]** must contain  $f(\mathbf{x})$  and  $f(\mathbf{y})$  respectively.

*On intermediate exit:* contains 2, 3 or 4. The calling program must evaluate  $f$  at  $\mathbf{x}$ , storing the result in  $\mathbf{fx}$ , and re-enter `nag_zero_cont_func_brent_rcomm` (c05azc) with all other arguments unchanged.

*On final exit:* contains 0.

*Constraint:* on entry  $\mathbf{ind} = -1, 1, 2, 3$  or 4.

- 8: **fail** – NagError \* *Input/Output*  
The NAG error argument (see Section 3.6 in the Essential Introduction).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_INT

On entry,  $\mathbf{ind} = \langle value \rangle$ .

Constraint:  $\mathbf{ind} = -1, 1, 2, 3$  or 4.

### 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 the Essential Introduction 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 the Essential Introduction for further information.

### NE\_NOT\_SIGN\_CHANGE

On entry,  $f(\mathbf{x})$  and  $f(\mathbf{y})$  have the same sign with neither equalling 0.0:  $f(\mathbf{x}) = \langle value \rangle$  and  $f(\mathbf{y}) = \langle value \rangle$ .

### NE\_PROBABLE\_POLE

The final interval may contain a pole rather than a zero. Note that this error exit is not completely reliable: it may be taken in extreme cases when  $[\mathbf{x}, \mathbf{y}]$  contains a zero, or it may not be taken when  $[\mathbf{x}, \mathbf{y}]$  contains a pole. Both these cases occur most frequently when  $\mathbf{tolx}$  is large.

### NE\_REAL

On entry,  $\mathbf{tolx} = \langle value \rangle$ .

Constraint:  $\mathbf{tolx} > 0.0$ .

### NW\_TOO\_MUCH\_ACC\_REQUESTED

The tolerance  $\mathbf{tolx}$  has been set too small for the problem being solved. However, the values  $\mathbf{x}$  and  $\mathbf{y}$  returned may well be good approximations to the zero.  $\mathbf{tolx} = \langle value \rangle$ .

## 7 Accuracy

The accuracy of the final value  $\mathbf{x}$  as an approximation of the zero is determined by  $\mathbf{tolx}$  and  $\mathbf{ir}$  (see Section 5). A relative accuracy criterion ( $\mathbf{ir} = 2$ ) should not be used when the initial values  $\mathbf{x}$  and  $\mathbf{y}$  are of different orders of magnitude. In this case a change of origin of the independent variable may be appropriate. For example, if the initial interval  $[\mathbf{x}, \mathbf{y}]$  is transformed linearly to the interval  $[1, 2]$ , then the zero can be determined to a precise number of figures using an absolute ( $\mathbf{ir} = 1$ ) or relative ( $\mathbf{ir} = 2$ ) error test and the effect of the transformation back to the original interval can also be determined. Except for the accuracy check, such a transformation has no effect on the calculation of the zero.

## 8 Parallelism and Performance

Not applicable.

## 9 Further Comments

For most problems, the time taken on each call to `nag_zero_cont_func_brent_rcomm` (c05azc) will be negligible compared with the time spent evaluating  $f(x)$  between calls to `nag_zero_cont_func_brent_rcomm` (c05azc).

If the calculation terminates because  $f(\mathbf{x}) = 0.0$ , then on return  $\mathbf{y}$  is set to  $\mathbf{x}$ . (In fact,  $\mathbf{y} = \mathbf{x}$  on return only in this case and, possibly, when `fail.code = NW_TOO_MUCH_ACC_REQUESTED`.) There is no guarantee that the value returned in  $\mathbf{x}$  corresponds to a **simple** root and you should check whether it does. One way to check this is to compute the derivative of  $f$  at the point  $\mathbf{x}$ , preferably analytically, or, if this is not possible, numerically, perhaps by using a central difference estimate. If  $f'(\mathbf{x}) = 0.0$ , then  $\mathbf{x}$  must correspond to a multiple zero of  $f$  rather than a simple zero.

## 10 Example

This example calculates a zero of  $e^{-x} - x$  with an initial interval  $[0, 1]$ ,  $\mathbf{tolx} = 1.0e-5$  and a mixed error test.

### 10.1 Program Text

```
/* nag_zero_cont_func_brent_rcomm (c05azc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 9, 2009.
 */

#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       fx, tolx, x, y;
    Integer      ind;
    Nag_ErrorControl ir;
    /* Arrays */
    double       c[17];
    NagError     fail;

    INIT_FAIL(fail);

    printf("nag_zero_cont_func_brent_rcomm (c05azc) Example Program Results\n");
    printf("\n Iterations\n");

    tolx = 1e-05;
```

```

x = 0.0;
y = 1.0;
ir = Nag_Mixed;
ind = 1;
fx = 0.0;
/* nag_zero_cont_func_brent_rcomm (c05azc).
 * Locates a simple zero of a continuous function.
 * Reverse communication.
 */
while (ind != 0)
{
    nag_zero_cont_func_brent_rcomm(&x, &y, fx, tolx, ir, c, &ind, &fail);

    if (ind != 0)
    {
        fx = exp(-x) - x;
        printf(" x = %8.5f    fx = %13.4e    ind = %2"NAG_IFMT"\n", x, fx,
            ind);
    }
}

if (fail.code == NE_NOERROR)
{
    printf("\n Solution\n");
    printf(" x = %8.5f    y = %8.5f\n", x, y);
}
else
{
    printf("%s\n", fail.message);
    if (fail.code == NE_PROBABLE_POLE ||
        fail.code == NW_TOO_MUCH_ACC_REQUESTED)
    {
        printf(" x = %8.5f    y = %8.5f\n", x, y);
    }
    exit_status = 1;
    goto END;
}

END:
return exit_status;
}

```

## 10.2 Program Data

None.

## 10.3 Program Results

nag\_zero\_cont\_func\_brent\_rcomm (c05azc) Example Program Results

```

Iterations
x = 0.00000    fx = 1.0000e+00    ind = 2
x = 1.00000    fx = -6.3212e-01    ind = 3
x = 0.61270    fx = -7.0814e-02    ind = 4
x = 0.56707    fx = 1.1542e-04    ind = 4
x = 0.56714    fx = -9.4481e-07    ind = 4
x = 0.56713    fx = 1.4727e-05    ind = 4
x = 0.56714    fx = -9.4481e-07    ind = 4

```

```

Solution
x = 0.56714    y = 0.56713

```

---