NAG Library Function Document

nag_inteq_abel1_weak (d05bec)

1 Purpose

nag_inteq_abel1_weak (d05bec) computes the solution of a weakly singular nonlinear convolution Volterra–Abel integral equation of the first kind using a fractional Backward Differentiation Formulae (BDF) method.

2 Specification

```
#include <nag.h>
#include <nagd05.h>
void nag_inteq_abel1_weak (
     double (*ck)(double t, Nag_Comm *comm),
     double (*cf)(double t, Nag_Comm *comm),
     double (*cg)(double s, double y, Nag_Comm *comm),
     Nag_WeightMode wtmode, Integer iorder, double tlim, double tolnl,
     Integer nmesh, double yn[], double rwsav[], Integer lrwsav,
     Nag_Comm *comm, NagError *fail)
```

3 Description

nag_inteq_abel1_weak (d05bec) computes the numerical solution of the weakly singular convolution Volterra–Abel integral equation of the first kind

$$f(t) + \frac{1}{\sqrt{\pi}}\int_0^t \frac{k(t-s)}{\sqrt{t-s}} g(s, y(s))\, ds = 0, \quad 0 \leq t \leq T. \tag{1}$$

Note the constant $\frac{1}{\sqrt{\pi}}$ in (1). It is assumed that the functions involved in (1) are sufficiently smooth and if

$$f(t) = t^{\beta} w(t) \quad \text{with} \quad \beta > -\tfrac{1}{2} \text{ and } w(t) \text{ smooth,} \tag{2}$$

then the solution $y(t)$ is unique and has the form $y(t) = t^{\beta-1/2} z(t)$, (see Lubich (1987)). It is evident from (1) that $f(0) = 0$. You are required to provide the value of $y(t)$ at $t = 0$. If $y(0)$ is unknown, Section 9 gives a description of how an approximate value can be obtained.

The function uses a fractional BDF linear multi-step method selected by you to generate a family of quadrature rules (see nag_inteq_abel_weak_weights (d05byc)). The BDF methods available in nag_inteq_abel1_weak (d05bec) are of orders 4, 5 and 6 ($= p$ say). For a description of the theoretical and practical background related to these methods we refer to Lubich (1987) and to Baker and Derakhshan (1987) and Hairer *et al.* (1988) respectively.

The algorithm is based on computing the solution $y(t)$ in a step-by-step fashion on a mesh of equispaced points. The size of the mesh is given by $T/(N-1)$, N being the number of points at which the solution is sought. These methods require $2p-2$ starting values which are evaluated internally. The computation of the lag term arising from the discretization of (1) is performed by fast Fourier transform (FFT) techniques when $N > 32 + 2p - 1$, and directly otherwise. The function does not provide an error estimate and you are advised to check the behaviour of the solution with a different value of N. An option is provided which avoids the re-evaluation of the fractional weights when nag_inteq_abel1_weak (d05bec) is to be called several times (with the same value of N) within the same program with different functions.

4 References

Baker C T H and Derakhshan M S (1987) FFT techniques in the numerical solution of convolution equations *J. Comput. Appl. Math.* **20** 5–24

Gorenflo R and Pfeiffer A (1991) On analysis and discretization of nonlinear Abel integral equations of first kind *Acta Math. Vietnam* **16** 211–262

Hairer E, Lubich Ch and Schlichte M (1988) Fast numerical solution of weakly singular Volterra integral equations *J. Comput. Appl. Math.* **23** 87–98

Lubich Ch (1987) Fractional linear multistep methods for Abel–Volterra integral equations of the first kind *IMA J. Numer. Anal* **7** 97–106

5 Arguments

1: **ck** – function, supplied by the user *External Function*

ck must evaluate the kernel $k(t)$ of the integral equation (1).

The specification of **ck** is:

```
double ck (double t, Nag_Comm *comm)
```

1: **t** – double *Input*

On entry: t, the value of the independent variable.

2: **comm** – Nag_Comm *

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

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_inteq_abel1_weak (d05bec) you may allocate memory and initialize these pointers with various quantities for use by **ck** when called from nag_inteq_abel1_weak (d05bec) (see Section 3.2.1.1 in the Essential Introduction).

2: **cf** – function, supplied by the user *External Function*

cf must evaluate the function $f(t)$ in (1).

The specification of **cf** is:

```
double cf (double t, Nag_Comm *comm)
```

1: **t** – double *Input*

On entry: t, the value of the independent variable.

2: **comm** – Nag_Comm *

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

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_inteq_abel1_weak (d05bec) you may allocate memory and initialize these pointers with various quantities for use by **cf** when called from nag_inteq_abel1_weak (d05bec) (see Section 3.2.1.1 in the Essential Introduction).

3: **cg** – function, supplied by the user *External Function*

cg must evaluate the function $g(s, y(s))$ in (1).

The specification of **cg** is:

```
double cg (double s, double y, Nag_Comm *comm)
```

1: **s** – double *Input*

On entry: s, the value of the independent variable.

2: **y** – double *Input*

On entry: the value of the solution y at the point **s**.

3: **comm** – Nag_Comm *

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

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_inteq_abel1_weak (d05bec) you may allocate memory and initialize these pointers with various quantities for use by **cg** when called from nag_inteq_abel1_weak (d05bec) (see Section 3.2.1.1 in the Essential Introduction).

4: **wtmode** – Nag_WeightMode *Input*

On entry: if the fractional weights required by the method need to be calculated by the function then set **wtmode** = Nag_InitWeights.

If **wtmode** = Nag_ReuseWeights, the function assumes the fractional weights have been computed by a previous call and are stored in **rwsav**.

Constraint: **wtmode** = Nag_InitWeights or Nag_ReuseWeights.

Note: when nag_inteq_abel1_weak (d05bec) is re-entered with a value of **wtmode** = Nag_ReuseWeights, the values of **nmesh**, **iorder** and the contents of **rwsav** MUST NOT be changed

5: **iorder** – Integer *Input*

On entry: p, the order of the BDF method to be used.

Suggested value: **iorder** = 4.

Constraint: $4 \leq$ **iorder** ≤ 6.

6: **tlim** – double *Input*

On entry: the final point of the integration interval, T.

Constraint: **tlim** $> 10 \times$ ***machine precision***.

7: **tolnl** – double *Input*

On entry: the accuracy required for the computation of the starting value and the solution of the nonlinear equation at each step of the computation (see Section 9).

Suggested value: **tolnl** $= \sqrt{\epsilon}$ where ϵ is the ***machine precision***.

Constraint: **tolnl** $> 10 \times$ ***machine precision***.

8: **nmesh** – Integer *Input*

On entry: N, the number of equispaced points at which the solution is sought.

Constraint: **nmesh** $= 2^m + 2 \times$ **iorder** $- 1$, where $m \geq 1$.

9: **yn**[**nmesh**] – double *Input/Output*

On entry: **yn**[0] must contain the value of $y(t)$ at $t = 0$ (see Section 9).

On exit: **yn**$[i-1]$ contains the approximate value of the true solution $y(t)$ at the point $t = (i-1) \times h$, for $i = 1, 2, \ldots,$ **nmesh**, where $h =$ **tlim**/(**nmesh** $- 1$).

10: **rwsav**[**lrwsav**] – double *Communication Array*

On entry: if **wtmode** = Nag_ReuseWeights, **rwsav** must contain fractional weights computed by a previous call of nag_inteq_abel1_weak (d05bec) (see description of **wtmode**).

On exit: contains fractional weights which may be used by a subsequent call of nag_inteq_abel1_weak (d05bec).

11: **lrwsav** – Integer *Input*

On entry: the dimension of the array **rwsav**.

Constraint: **lrwsav** $\geq (2 \times$ **iorder** $+ 6) \times$ **nmesh** $+ 8 \times$ **iorder**$^2 - 16 \times$ **iorder** $+ 1$.

12: **comm** – Nag_Comm *

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

13: **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 ⟨*value*⟩ had an illegal value.

NE_FAILED_START

An error occurred when trying to compute the starting values.

NE_FAILED_STEP

An error occurred when trying to compute the solution at a specific step.

NE_INT

On entry, **iorder** = ⟨*value*⟩.
Constraint: $4 \leq$ **iorder** ≤ 6.

NE_INT_2

On entry, **lrwsav** = ⟨*value*⟩.
Constraint: **lrwsav** $\geq (2 \times$ **iorder** $+ 6) \times$ **nmesh** $+ 8 \times$ **iorder**$^2 - 16 \times$ **iorder** $+ 1$; that is, ⟨*value*⟩.

On entry, **nmesh** = ⟨*value*⟩ and **iorder** = ⟨*value*⟩.
Constraint: **nmesh** $= 2^m + 2 \times$ **iorder** $- 1$, for some m.

On entry, **nmesh** $= \langle value \rangle$ and **iorder** $= \langle value \rangle$.
Constraint: **nmesh** $\geq 2 \times$ **iorder** $+ 1$.

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_REAL

On entry, **tlim** $= \langle value \rangle$.
Constraint: **tlim** $> 10 \times$ ***machine precision***

On entry, **tolnl** $= \langle value \rangle$.
Constraint: **tolnl** $> 10 \times$ ***machine precision***.

7 Accuracy

The accuracy depends on **nmesh** and **tolnl**, the theoretical behaviour of the solution of the integral equation and the interval of integration. The value of **tolnl** controls the accuracy required for computing the starting values and the solution of (3) at each step of computation. This value can affect the accuracy of the solution. However, for most problems, the value of $\sqrt{\epsilon}$, where ϵ is the ***machine precision***, should be sufficient.

8 Parallelism and Performance

nag_inteq_abel1_weak (d05bec) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_inteq_abel1_weak (d05bec) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

Also when solving (1) the initial value $y(0)$ is required. This value may be computed from the limit relation (see Gorenflo and Pfeiffer (1991))

$$\frac{-2}{\sqrt{\pi}} k(0) g(0, y(0)) = \lim_{t \to 0} \frac{f(t)}{\sqrt{t}}. \tag{3}$$

If the value of the above limit is known then by solving the nonlinear equation (3) an approximation to $y(0)$ can be computed. If the value of the above limit is not known, an approximation should be provided. Following the analysis presented in Gorenflo and Pfeiffer (1991), the following pth-order approximation can be used:

$$\lim_{t \to 0} \frac{f(t)}{\sqrt{t}} \simeq \frac{f(h^p)}{h^{p/2}}. \tag{4}$$

However, it must be emphasized that the approximation in (4) may result in an amplification of the rounding errors and hence you are advised (if possible) to determine $\lim_{t \to 0} \frac{f(t)}{\sqrt{t}}$ by analytical methods.

Also when solving (1), initially, nag_inteq_abel1_weak (d05bec) computes the solution of a system of nonlinear equation for obtaining the $2p - 2$ starting values. nag_zero_nonlin_eqns_rcomm (c05qdc) is used for this purpose. If a failure with **fail**.**code** = NE_FAILED_START occurs (corresponding to an error exit from nag_zero_nonlin_eqns_rcomm (c05qdc)), you are advised to either relax the value of **tolnl** or choose a smaller step size by increasing the value of **nmesh**. Once the starting values are computed successfully, the solution of a nonlinear equation of the form

$$Y_n - \alpha g(t_n, Y_n) - \Psi_n = 0, \qquad (5)$$

is required at each step of computation, where Ψ_n and α are constants. nag_inteq_abel1_weak (d05bec) calls nag_zero_cont_func_cntin_rcomm (c05axc) to find the root of this equation.

When a failure with **fail**.**code** = NE_FAILED_STEP occurs (which corresponds to an error exit from nag_zero_cont_func_cntin_rcomm (c05axc)), you are advised to either relax the value of the **tolnl** or choose a smaller step size by increasing the value of **nmesh**.

If a failure with **fail**.**code** = NE_FAILED_START or NE_FAILED_STEP persists even after adjustments to **tolnl** and/or **nmesh** then you should consider whether there is a more fundamental difficulty. For example, the problem is ill-posed or the functions in (1) are not sufficiently smooth.

10 Example

We solve the following integral equations.

Example 1

The density of the probability that a Brownian motion crosses a one-sided moving boundary $a(t)$ before time t, satisfies the integral equation (see Hairer *et al.* (1988))

$$-\frac{1}{\sqrt{t}} \exp\left(\tfrac{1}{2} - \{a(t)\}^2/t\right) + \int_0^t \frac{\exp\left(-\frac{1}{2}\{a(t) - a(s)\}^2/(t-s)\right)}{\sqrt{t-s}} y(s)\,ds = 0, \quad 0 \le t \le 7.$$

In the case of a straight line $a(t) = 1 + t$, the exact solution is known to be

$$y(t) = \frac{1}{\sqrt{2\pi t^3}} \exp\left\{-(1+t)^2/2t\right\}$$

Example 2

In this example we consider the equation

$$-\frac{2\log\left(\sqrt{1+t} + \sqrt{t}\right)}{\sqrt{1+t}} + \int_0^t \frac{y(s)}{\sqrt{t-s}} ds = 0, \quad 0 \le t \le 5.$$

The solution is given by $y(t) = \frac{1}{1+t}$.

In the above examples, the fourth-order BDF is used, and **nmesh** is set to $2^6 + 7$.

10.1 Program Text

```
/* nag_inteq_abel1_weak (d05bec) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd05.h>
#include <nagx01.h>
#include <nagx02.h>
```

```
#ifdef __cplusplus
extern "C" {
#endif
  static double NAG_CALL ck1(double t, Nag_Comm *comm);
  static double NAG_CALL cf1(double t, Nag_Comm *comm);
  static double NAG_CALL cg1(double s, double y, Nag_Comm *comm);
  static double NAG_CALL ck2(double t, Nag_Comm *comm);
  static double NAG_CALL cf2(double t, Nag_Comm *comm);
  static double NAG_CALL cg2(double s, double y, Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

static double sol1(double t);
static double sol2(double t);

int main(void)
{
  /* Scalars */
  double  err, errmax, h, hi1, soln, t, tlim, tolnl;
  Integer exit_status = 0;
  Integer iorder = 4;
  Integer i, iskip, exno, nmesh, lrwsav;
  /* Arrays */
  static double ruser[6] = {-1.0, -1.0, -1.0, -1.0, -1.0, -1.0};
  double  *rwsav = 0, *yn = 0;
  /* NAG types */
  Nag_Comm comm;
  NagError fail;
  Nag_WeightMode wtmode;

  INIT_FAIL(fail);

  printf("nag_inteq_abel1_weak (d05bec) Example Program Results\n");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  nmesh = pow(2, 6) + 7;
  lrwsav = (2 * iorder + 6) * nmesh + 8 * pow(iorder, 2) - 16 * iorder + 1;

  if (
      !(yn = NAG_ALLOC(nmesh, double)) ||
      !(rwsav = NAG_ALLOC(lrwsav, double))
      )
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  tolnl = sqrt(nag_machine_precision);

  for (exno = 1; exno <= 2; exno++)
    {
      printf("\nExample %"NAG_IFMT"\n\n", exno);

      if (exno==1)
        {
          tlim = 7.0;
          iskip = 5;
          h = tlim/(double) (nmesh - 1);
          wtmode = Nag_InitWeights;
          yn[0] = 0.0;

          /*
            nag_inteq_abel1_weak (d05bec).
            Nonlinear convolution Volterra-Abel equation, first kind,
            weakly singular.
          */
```

```
          nag_inteq_abel1_weak(ck1, cf1, cg1, wtmode, iorder, tlim, tolnl,
                               nmesh, yn, rwsav, lrwsav, &comm, &fail);
        }
      else
        {
          tlim = 5.0;
          iskip = 7;
          h = tlim/(double) (nmesh - 1);
          wtmode = Nag_ReuseWeights;
          yn[0] = 1.0;

          /* nag_inteq_abel1_weak (d05bec) as above. */
          nag_inteq_abel1_weak(ck2, cf2, cg2, wtmode, iorder, tlim, tolnl,
                               nmesh, yn, rwsav, lrwsav, &comm, &fail);
        }
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_inteq_abel1_weak (d05bec).\n%s\n",
                 fail.message);
          exit_status = 1;
          goto END;
        }

      printf("The stepsize h = %8.4f\n\n", h);
      printf("     t        Approximate\n");
      printf("                Solution\n\n");
      errmax = 0.0;

      for (i = 0; i < nmesh; i++)
        {
          hi1 = (double) (i) * h;
          err = fabs(yn[i] - ((exno==1)?sol1(hi1):sol2(hi1)));

          if (err > errmax)
            {
              errmax = err;
              t = hi1;
              soln = yn[i];
            }

          if (i > 0 && i%iskip == 0) printf("%8.4f%15.4f\n", hi1, yn[i]);
        }

      printf("\nThe maximum absolute error, %10.2e, occurred at t = %8.4f\n",
             errmax, t);
      printf("with solution %8.4f\n", soln);
    }

 END:
  NAG_FREE(rwsav);
  NAG_FREE(yn);

  return exit_status;
}

static double sol1(double t)
{
  return (1.0 / (sqrt(2.0 * nag_pi) * pow(t, 1.5))) *
    exp(-pow(1.0 + t, 2) / (2.0 * t));
}
static double sol2(double t)
{
  return 1.0/(1.0 + t);
}
static double NAG_CALL ck1(double t, Nag_Comm *comm)
{
  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback ck1, first invocation.)\n");
      comm->user[0] = 0.0;
    }
```

```
  return exp(-0.5 * t);
}
static double NAG_CALL cf1(double t, Nag_Comm *comm)
{
  if (comm->user[1] == -1.0)
    {
      printf("(User-supplied callback cf1, first invocation.)\n");
      comm->user[1] = 0.0;
    }
  return (-1.0 / sqrt(nag_pi * t)) * exp(-0.5 * pow(1.0 + t, 2) / t);
}
static double NAG_CALL cg1(double s, double y, Nag_Comm *comm)
{
  if (comm->user[2] == -1.0)
    {
      printf("(User-supplied callback cg1, first invocation.)\n");
      comm->user[2] = 0.0;
    }
  return y;
}
static double NAG_CALL ck2(double t, Nag_Comm *comm)
{
  if (comm->user[3] == -1.0)
    {
      printf("(User-supplied callback ck2, first invocation.)\n");
      comm->user[3] = 0.0;
    }
  return sqrt(nag_pi);
}
static double NAG_CALL cf2(double t, Nag_Comm *comm)
{
  /* Scalars */
  double st1;

  if (comm->user[4] == -1.0)
    {
      printf("(User-supplied callback cf2, first invocation.)\n");
      comm->user[4] = 0.0;
    }
  st1 = sqrt(1.0 + t);
  return -2.0 * log(st1 + sqrt(t)) / st1;
}
static double NAG_CALL cg2(double s, double y, Nag_Comm *comm)
{
  if (comm->user[5] == -1.0)
    {
      printf("(User-supplied callback cg2, first invocation.)\n");
      comm->user[5] = 0.0;
    }
  return y;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_inteq_abel1_weak (d05bec) Example Program Results

Example 1

(User-supplied callback ck1, first invocation.)
(User-supplied callback cf1, first invocation.)
(User-supplied callback cg1, first invocation.)
The stepsize h =   0.1000

     t        Approximate
                Solution
```

```
  0.5000         0.1191
  1.0000         0.0528
  1.5000         0.0265
  2.0000         0.0146
  2.5000         0.0086
  3.0000         0.0052
  3.5000         0.0033
  4.0000         0.0022
  4.5000         0.0014
  5.0000         0.0010
  5.5000         0.0007
  6.0000         0.0004
  6.5000         0.0003
  7.0000         0.0002

The maximum absolute error,   2.86e-03, occurred at t =   0.1000
with solution   0.0326

Example 2

(User-supplied callback ck2, first invocation.)
(User-supplied callback cf2, first invocation.)
(User-supplied callback cg2, first invocation.)
The stepsize h =   0.0714

     t        Approximate
                Solution

  0.5000         0.6667
  1.0000         0.5000
  1.5000         0.4000
  2.0000         0.3333
  2.5000         0.2857
  3.0000         0.2500
  3.5000         0.2222
  4.0000         0.2000
  4.5000         0.1818
  5.0000         0.1667

The maximum absolute error,   3.17e-06, occurred at t =   0.0714
with solution   0.9333
```