NAG Library Function Document

nag_inteq_volterra2 (d05bac)

1 Purpose

nag_inteq_volterra2 (d05bac) computes the solution of a nonlinear convolution Volterra integral equation of the second kind using a reducible linear multi-step method.

2 Specification

```
#include <nag.h>
#include <nagd05.h>
void nag_inteq_volterra2 (
     double (*ck)(double t, Nag_Comm *comm),
     double (*cg)(double s, double y, Nag_Comm *comm),
     double (*cf)(double t, Nag_Comm *comm),
     Nag_ODEMethod method, Integer iorder, double alim, double tlim,
     double tol, Integer nmesh, double thresh, double work[], Integer lwk,
     double yn[], double errest[], Nag_Comm *comm, NagError *fail)
```

3 Description

nag_inteq_volterra2 (d05bac) computes the numerical solution of the nonlinear convolution Volterra integral equation of the second kind

$$y(t) = f(t) + \int_a^t k(t-s)g(s,y(s))\,ds, \quad a \leq t \leq T. \tag{1}$$

It is assumed that the functions involved in (1) are sufficiently smooth. The function uses a reducible linear multi-step formula selected by you to generate a family of quadrature rules. The reducible formulae available in nag_inteq_volterra2 (d05bac) are the Adams–Moulton formulae of orders 3 to 6, and the backward differentiation formulae (BDF) of orders 2 to 5. For more information about the behaviour and the construction of these rules we refer to Lubich (1983) and Wolkenfelt (1982).

The algorithm is based on computing the solution in a step-by-step fashion on a mesh of equispaced points. The initial step size which is given by $(T-a)/N$, N being the number of points at which the solution is sought, is halved and another approximation to the solution is computed. This extrapolation procedure is repeated until successive approximations satisfy a user-specified error requirement.

The above methods require some starting values. For the Adams' formula of order greater than 3 and the BDF of order greater than 2 we employ an explicit Dormand–Prince–Shampine Runge–Kutta method (see Shampine (1986)). The above scheme avoids the calculation of the kernel, $k(t)$, on the negative real line.

4 References

Lubich Ch (1983) On the stability of linear multi-step methods for Volterra convolution equations *IMA J. Numer. Anal.* **3** 439–465

Shampine L F (1986) Some practical Runge–Kutta formulas *Math. Comput.* **46(173)** 135–150

Wolkenfelt P H M (1982) The construction of reducible quadrature rules for Volterra integral and integro-differential equations *IMA J. Numer. Anal.* **2** 131–152

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_volterra2 (d05bac) you may allocate memory and initialize these pointers with various quantities for use by **ck** when called from nag_inteq_volterra2 (d05bac) (see Section 3.2.1.1 in the Essential Introduction).

2: **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_volterra2 (d05bac) you may allocate memory and initialize these pointers with various quantities for use by **cg** when called from nag_inteq_volterra2 (d05bac) (see Section 3.2.1.1 in the Essential Introduction).

3: **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_volterra2 (d05bac) you may allocate memory and initialize these pointers with various quantities for use by **cf** when called from nag_inteq_volterra2 (d05bac) (see Section 3.2.1.1 in the Essential Introduction).

4: **method** – Nag_ODEMethod *Input*

On entry: the type of method which you wish to employ.

method = Nag_Adams
For Adams' type formulae.

method = Nag_BDF
For backward differentiation formulae.

Constraint: **method** = Nag_Adams or Nag_BDF.

5: **iorder** – Integer *Input*

On entry: the order of the method to be used.

Constraints:

if **method** = Nag_Adams, $3 \leq$ **iorder** ≤ 6;
if **method** = Nag_BDF, $2 \leq$ **iorder** ≤ 5.

6: **alim** – double *Input*

On entry: a, the lower limit of the integration interval.

Constraint: **alim** ≥ 0.0.

7: **tlim** – double *Input*

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

Constraint: **tlim** $>$ **alim**.

8: **tol** – double *Input*

On entry: the relative accuracy required in the computed values of the solution.

Constraint: $\sqrt{\epsilon} \leq$ **tol** ≤ 1.0, where ϵ is the ***machine precision***.

9: **nmesh** – Integer *Input*

On entry: the number of equidistant points at which the solution is sought.

Constraints:

if **method** = Nag_Adams, **nmesh** $\geq$ **iorder** $- 1$;
if **method** = Nag_BDF, **nmesh** $\geq$ **iorder**.

10: **thresh** – double *Input*

On entry: the threshold value for use in the evaluation of the estimated relative errors. For two successive meshes the following condition must hold at each point of the coarser mesh

$$\frac{|Y_1 - Y_2|}{\max(|Y_1|, |Y_2|, |\mathbf{thresh}|)} \leq \mathbf{tol},$$

where Y_1 is the computed solution on the coarser mesh and Y_2 is the computed solution at the corresponding point in the finer mesh. If this condition is not satisfied then the step size is halved and the solution is recomputed.

Note: **thresh** can be used to effect a relative, absolute or mixed error test. If **thresh** $= 0.0$ then pure relative error is measured and, if the computed solution is small and **thresh** $= 1.0$, absolute error is measured.

11: **work**[**lwk**] – double *Output*

12: **lwk** – Integer *Input*

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

Constraint: **lwk** $\geq 10 \times$ **nmesh** $+ 6$.

Note: the above value of **lwk** is sufficient for nag_inteq_volterra2 (d05bac) to perform only one extrapolation on the initial mesh as defined by **nmesh**. In general much more workspace is required and in the case when a large step size is supplied (i.e., **nmesh** is small), you must provide a considerably larger workspace.

On exit: if **fail.code** = NW_OUT_OF_WORKSPACE, **work**[0] contains the size of **lwk** required for the algorithm to proceed further.

13: **yn**[**nmesh**] – double *Output*

On exit: **yn**$[i-1]$ contains the most recent approximation of the true solution $y(t)$ at the specified point $t = \mathbf{alim} + i \times H$, for $i = 1, 2, \ldots, \mathbf{nmesh}$, where $H = (\mathbf{tlim} - \mathbf{alim})/\mathbf{nmesh}$.

14: **errest**[**nmesh**] – double *Output*

On exit: **errest**$[i-1]$ contains the most recent approximation of the relative error in the computed solution at the point $t = \mathbf{alim} + i \times H$, for $i = 1, 2, \ldots, \mathbf{nmesh}$, where $H = (\mathbf{tlim} - \mathbf{alim})/\mathbf{nmesh}$.

15: **comm** – Nag_Comm *

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

16: **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_CONVERGENCE

The solution is not converging. See Section 9.

NE_ENUM_INT

On entry, **method** $=$ Nag_Adams and **iorder** $= 2$.
Constraint: if **method** $=$ Nag_Adams, $3 \leq$ **iorder** ≤ 6.

On entry, **method** $=$ Nag_BDF and **iorder** $= 6$.
Constraint: if **method** $=$ Nag_BDF, $2 \leq$ **iorder** ≤ 5.

NE_ENUM_INT_2

On entry, **method** $=$ Nag_Adams, **iorder** $= \langle value \rangle$ and **nmesh** $= \langle value \rangle$.
Constraint: if **method** $=$ Nag_Adams, **nmesh** $\geq$ **iorder** $- 1$.

On entry, **method** $=$ Nag_BDF, **iorder** $= \langle value \rangle$ and **nmesh** $= \langle value \rangle$.
Constraint: if **method** $=$ Nag_BDF, **nmesh** $\geq$ **iorder**.

NE_INT

On entry, **iorder** $= \langle value \rangle$.
Constraint: $2 \leq$ **iorder** ≤ 6.

On entry, **lwk** $= \langle value \rangle$.
Constraint: **lwk** $\geq 10 \times$ **nmesh** $+ 6$; that is, $\langle value \rangle$.

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, **alim** $= \langle value \rangle$.
Constraint: **alim** ≥ 0.0.

On entry, **tol** $= \langle value \rangle$.
Constraint: $\sqrt{\textit{\textbf{machine precision}}} \leq$ **tol** ≤ 1.0.

NE_REAL_2

On entry, **alim** $= \langle value \rangle$ and **tlim** $= \langle value \rangle$.
Constraint: **tlim** $>$ **alim**.

NW_OUT_OF_WORKSPACE

The workspace which has been supplied is too small for the required accuracy. The number of extrapolations, so far, is $\langle value \rangle$. If you require one more extrapolation extend the size of workspace to: **lwk** $= \langle value \rangle$.

7 Accuracy

The accuracy depends on **tol**, the theoretical behaviour of the solution of the integral equation, the interval of integration and on the method being used. It can be controlled by varying **tol** and **thresh**; you are recommended to choose a smaller value for **tol**, the larger the value of **iorder**.

You are warned not to supply a very small **tol**, because the required accuracy may never be achieved. This will usually force an error exit with **fail.code** $=$ NW_OUT_OF_WORKSPACE.

In general, the higher the order of the method, the faster the required accuracy is achieved with less workspace. For non-stiff problems (see Section 9) you are recommended to use the Adams' method (**method** $=$ Nag_Adams) of order greater than 4 (**iorder** > 4).

8 Parallelism and Performance

Not applicable.

9 Further Comments

When solving (1), the solution of a nonlinear equation of the form

$$Y_n - \alpha g(t_n, Y_n) - \Psi_n = 0, \tag{2}$$

is required, where Ψ_n and α are constants. nag_inteq_volterra2 (d05bac) calls nag_interval_zero_cont_func (c05avc) to find an interval for the zero of this equation followed by nag_zero_cont_func_brent_rcomm (c05azc) to find its zero.

There is an initial phase of the algorithm where the solution is computed only for the first few points of the mesh. The exact number of these points depends on **iorder** and **method**. The step size is halved until the accuracy requirements are satisfied on these points and only then the solution on the whole mesh is computed. During this initial phase, if **lwk** is too small, nag_inteq_volterra2 (d05bac) will exit with **fail.code** $=$ NW_OUT_OF_WORKSPACE.

In the case **fail.code** $=$ NE_CONVERGENCE or NW_OUT_OF_WORKSPACE, you may be dealing with a 'stiff' equation; an equation where the Lipschitz constant L of the function $g(t,y)$ in (1) with respect to its second argument is large, viz,

$$|g(t,u) - g(t,v)| \leq L|u-v|. \tag{3}$$

In this case, if a BDF method (**method** $=$ Nag_BDF) has been used, you are recommended to choose a smaller step size by increasing the value of **nmesh**, or provide a larger workspace. But, if an Adams' method (**method** $=$ Nag_Adams) has been selected, you are recommended to switch to a BDF method instead.

In the case **fail.code** $=$ NW_OUT_OF_WORKSPACE, then if **fail.errnum** $= 6$, the specified accuracy has not been attained but **yn** and **errest** contain the most recent approximation to the computed solution and the corresponding error estimate. In this case, the error message informs you of the number of extrapolations performed and the size of **lwk** required for the algorithm to proceed further. The latter quantity will also be available in **work**[0].

10 Example

Consider the following integral equation

$$y(t) = e^{-t} + \int_0^t e^{-(t-s)} \left[y(s) + e^{-y(s)} \right] ds, \quad 0 \leq t \leq 20 \tag{4}$$

with the solution $y(t) = \ln(t+e)$. In this example, the Adams' method of order 6 is used to solve this equation with **tol** $=$ 1.e$-$4.

10.1 Program Text

```
/* nag_inteq_volterra2 (d05bac) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd05.h>
#include <nagx02.h>
```

```
#ifdef __cplusplus
extern "C" {
#endif

  static double NAG_CALL sol(double t);
  static double NAG_CALL cf(double t, Nag_Comm *comm);
  static double NAG_CALL ck(double t, Nag_Comm *comm);
  static double NAG_CALL cg(double s, double y, Nag_Comm *comm);

#ifdef __cplusplus
}
#endif

int main(void)
{
  /* Scalars */
  double      alim = 0.0, tlim = 20.0, tol = 1.e-4;
  double      h, hi, si, thresh;
  Integer     exit_status = 0;
  Integer     iorder = 6, nmesh = 6;
  Integer     i, lwk;
  /* Arrays */
  static double ruser[3] = {-1.0, -1.0, -1.0};
  double      *errest = 0, *work = 0, *yn = 0;
  /* NAG types */
  Nag_Comm comm;
  NagError fail;
  Nag_ODEMethod method = Nag_Adams;

  INIT_FAIL(fail);

  printf("nag_inteq_volterra2 (d05bac) Example Program Results\n");

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

  lwk = 10 * nmesh + 6;

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

  h = (tlim - alim)/(double) (nmesh);
  thresh = nag_machine_precision;


  /*
    nag_inteq_volterra2 (d05bac).
    Nonlinear Volterra convolution equation, second kind.
  */
  nag_inteq_volterra2(ck, cg, cf, method, iorder, alim, tlim, tol, nmesh,
                      thresh, work, lwk, yn, errest, &comm, &fail);
  /* Loop until the supplied workspace is big enough. */
  while (fail.code == NW_OUT_OF_WORKSPACE)
    {
      lwk = work[0];
      NAG_FREE(work);

      if (!(work = NAG_ALLOC(lwk, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
```

```
        }
      nag_inteq_volterra2(ck, cg, cf, method, iorder, alim, tlim, tol, nmesh,
                          thresh, work, lwk, yn, errest, &comm, &fail);
    }

  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_inteq_volterra2 (d05bac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  printf("\nSize of workspace = %"NAG_IFMT"\n", lwk);
  printf("Tolerance         = %f\n\n", tol);
  printf("   t        Approx. Sol.  True Sol.    Est. Error    Actual Error\n");

  hi = 0.0;
  for (i = 0; i < nmesh; i++)
    {
      hi += h;
      si = sol(hi);
      printf("%7.2f%14.5f%14.5f%15.5e%15.5e\n", alim + hi, yn[i], si,
              errest[i], fabs((yn[i] - si)/si));
    }

 END:

  NAG_FREE(errest);
  NAG_FREE(yn);
  NAG_FREE(work);

  return exit_status;
}

static double NAG_CALL sol(double t)
{
  return log(t + exp(1.0));
}
static double NAG_CALL cf(double t, Nag_Comm *comm)
{
  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback cf, first invocation.)\n");
      comm->user[0] = 0.0;
    }
  return exp(-t);
}
static double NAG_CALL ck(double t, Nag_Comm *comm)
{
  if (comm->user[1] == -1.0)
    {
      printf("(User-supplied callback ck, first invocation.)\n");
      comm->user[1] = 0.0;
    }
  return exp(-t);
}
static double NAG_CALL cg(double s, double y, Nag_Comm *comm)
{
  if (comm->user[2] == -1.0)
    {
      printf("(User-supplied callback cg, first invocation.)\n");
      comm->user[2] = 0.0;
    }
  return y + exp(-y);
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_inteq_volterra2 (d05bac) Example Program Results
(User-supplied callback ck, first invocation.)
(User-supplied callback cf, first invocation.)
(User-supplied callback cg, first invocation.)

Size of workspace = 966
Tolerance         = 0.000100

   t        Approx. Sol.  True Sol.    Est. Error    Actual Error
   3.33       1.80033       1.80033    4.46315e-07    1.86622e-06
   6.67       2.23911       2.23911    2.14707e-06    3.39762e-06
  10.00       2.54305       2.54304    2.49406e-06    3.48516e-06
  13.33       2.77582       2.77581    6.47098e-06    3.31131e-06
  16.67       2.96451       2.96450    8.91042e-06    3.09688e-06
  20.00       3.12318       3.12317    1.08231e-05    2.89422e-06
```