NAG Library Function Document

nag_inteq_volterra_weights (d05bwc)

1 Purpose

nag_inteq_volterra_weights (d05bwc) computes the quadrature weights associated with the Adams methods of orders three to six and the Backward Differentiation Formulae (BDF) methods of orders two to five. These rules, which are referred to as reducible quadrature rules, can then be used in the solution of Volterra integral and integro-differential equations.

2 Specification

```
#include <nag.h>
#include <nagd05.h>
void nag_inteq_volterra_weights (Nag_ODEMethod method, Integer iorder,
     Integer nomg, double omega[], double sw[], NagError *fail)
```

3 Description

nag_inteq_volterra_weights (d05bwc) computes the weights $W_{i,j}$ and ω_i for a family of quadrature rules related to the Adams methods of orders three to six and the BDF methods of orders two to five, for approximating the integral:

$$\int_0^t \phi(s)\,ds \simeq h\sum_{j=0}^{p-1} W_{i,j}\phi(j\times h) + h\sum_{j=p}^{i}\omega_{i-j}\phi(j\times h), \qquad 0 \le t \le T, \tag{1}$$

with $t = i \times h$, for $i = 0, 1, \ldots, n$, for some given constant h.

In (1), h is a uniform mesh, p is related to the order of the method being used and $W_{i,j}$, ω_i are the starting and the convolution weights respectively. The mesh size h is determined as $h = \frac{T}{n}$, where $n = n_w + p - 1$ and n_w is the chosen number of convolution weights w_j, for $j = 1, 2, \ldots, n_w - 1$. A description of how these weights can be used in the solution of a Volterra integral equation of the second kind is given in Section 8. For a general discussion of these methods, see Wolkenfelt (1982) for more details.

4 References

Lambert J D (1973) *Computational Methods in Ordinary Differential Equations* John Wiley

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: **method** – Nag_ODEMethod *Input*

On entry: the type of method to be used.

method = Nag_Adams
For Adams type formulae.

method = Nag_BDF
For Backward Differentiation Formulae.

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

2: **iorder** – Integer *Input*

On entry: the order of the method to be used. The number of starting weights, p is determined by **method** and **iorder**.

If **method** = Nag_Adams, $p =$ **iorder** $- 1$.

If **method** = Nag_BDF, $p =$ **iorder**.

Constraints:

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

3: **nomg** – Integer *Input*

On entry: the number of convolution weights, n_w.

Constraint: **nomg** ≥ 1.

4: **omega**[**nomg**] – double *Output*

On exit: contains the first **nomg** convolution weights.

5: **sw**[$n \times p$] – double *Output*

On exit: **sw**[$j \times n + i - 1$] contains the weights $W_{i,j}$, for $i = 1, 2, \ldots, n$ and $j = 0, 1, \ldots, p - 1$, where n is as defined in Section 3.

6: **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_ENUM_INT

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

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

NE_INT

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

On entry, **nomg** = ⟨*value*⟩.
Constraint: **nomg** ≥ 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.

7 Accuracy

Not applicable.

8 Further Comments

Reducible quadrature rules are most appropriate for solving Volterra integral equations (and integro-differential equations). In this section, we propose the following algorithm which you may find useful in solving a linear Volterra integral equation of the form

$$y(t) = f(t) + \int_0^t K(t,s)y(s)\,ds, \qquad 0 \le t \le T, \tag{2}$$

using nag_inteq_volterra_weights (d05bwc). In (2), $K(t,s)$ and $f(t)$ are given and the solution $y(t)$ is sought on a uniform mesh of size h such that $T = nh$. Discretization of (2) yields

$$y_i = f(i \times h) + h\sum_{j=0}^{p-1} W_{i,j}K(i,h,j,h)y_j + h\sum_{j=p}^{i} \omega_{i-j}K(i,h,j,h)y_j, \tag{3}$$

where $y_i \simeq y(i \times h)$. We propose the following algorithm for computing y_i from (3) after a call to nag_inteq_volterra_weights (d05bwc):

(a) Equation (3) requires starting values, y_j, for $j = 1, 2, \ldots, p-1$, with $y_0 = f(0)$. These starting values can be computed by solving the linear system

$$y_i = f(i \times h) + h\sum_{j=0}^{p-1} \mathbf{sw}[j \times n + i - 1]K(i,h,j,h)y_j, \qquad i = 1, 2, \ldots, p-1.$$

(b) Compute the inhomogeneous terms

$$\sigma_i = f(i \times h) + h\sum_{j=0}^{p-1} \mathbf{sw}[j \times n + i - 1]K(i,h,j,h)y_j, \qquad i = p, p+1, \ldots, n.$$

(c) Start the iteration for $i = p, p+1, \ldots, n$ to compute y_i from:

$$(1 - h \times \mathbf{omega}[0]K(i,h,i,h))y_i = \sigma_i + h\sum_{j=p}^{i-1} \mathbf{omega}[i-j]K(i,h,j,h)y_j.$$

Note that for a nonlinear integral equation, the solution of a nonlinear algebraic system is required at step (a) and a single nonlinear equation at step (c).

9 Example

The following example generates the first ten convolution and thirteen starting weights generated by the fourth-order BDF method.

9.1 Program Text

```
/* nag_inteq_volterra_weights (d05bwc) Example Program.
 *
 * Copyright 2011, Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd05.h>

int main(void)
{
  /* Scalars */
  Integer exit_status = 0;
  Integer i, iorder, j, nomg, p, n;
  char    methodstring[10];
  /* Arrays */
  double  *omega = 0, *sw = 0;
  /* NAG types */
  NagError fail;
  Nag_ODEMethod method;

  INIT_FAIL(fail);

  printf("nag_inteq_volterra_weights (d05bwc) Example Program Results\n");
```

```
  /* Skip heading in data file*/
  scanf("%*[^\n] ");
  scanf("%s%*[^\n] ", methodstring);

  /*
   * nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value.
   */
  method = (Nag_ODEMethod) nag_enum_name_to_value(methodstring);

  scanf("%ld%*[^\n] ", &iorder);
  scanf("%ld%*[^\n] ", &nomg);

  switch (method)
    {
    case Nag_Adams:
      p = iorder - 1;
      break;
    case Nag_BDF:
      p = iorder;
      break;
    }

  n = nomg + p - 1;

  if (
      !(omega = NAG_ALLOC(nomg, double)) ||
      !(sw = NAG_ALLOC(p*n, double))
      )
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /*
    nag_inteq_volterra_weights (d05bwc).
    Generate weights for use in solving Volterra equations.
  */
  nag_inteq_volterra_weights(method, iorder, nomg, omega, sw, &fail);

  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_inteq_volterra_weights (d05bwc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  printf("\nThe convolution weights\n\n  n-j        omega\n");
  for (j = 0; j < nomg; j++)
    printf("%3ld     %10.4f\n", j+1, omega[j]);

  printf("\nThe weights W\n");
  printf("\n  i ");
  for (j = 0; j < p; j++)
    printf("%11s%ld ","j = ",j);
  printf("\n");

#define SW(I, J) sw[J * n + I]

  for (i = 0; i < n; i++)
    {
      printf("%3ld", i+1);
      for (j = 0; j < p; j++) printf("%13.4f", SW(i, j));
      printf("\n");
    }

#undef SW
```

```
 END:

  if (sw) NAG_FREE(sw);
  if (omega) NAG_FREE(omega);

  return exit_status;
}
```

9.2 Program Data

```
nag_inteq_volterra_weights (d05bwc) Example Program Data
  Nag_BDF                                                  : method
  4                                                        : iorder
  10                                                       : nomg
```

9.3 Program Results

```
nag_inteq_volterra_weights (d05bwc) Example Program Results

The convolution weights

  n-j        omega
  1         0.4800
  2         0.9216
  3         1.0783
  4         1.0504
  5         0.9962
  6         0.9797
  7         0.9894
  8         1.0003
  9         1.0034
 10         1.0017

The weights W

  i        j = 0        j = 1        j = 2        j = 3
  1       0.3750       0.7917      -0.2083       0.0417
  2       0.3333       1.3333       0.3333       0.0000
  3       0.3750       1.1250       1.1250       0.3750
  4       0.4800       0.7467       1.5467       0.7467
  5       0.5499       0.5719       1.5879       0.8886
  6       0.5647       0.5829       1.5016       0.8709
  7       0.5545       0.6385       1.4514       0.8254
  8       0.5458       0.6629       1.4550       0.8098
  9       0.5449       0.6578       1.4741       0.8170
 10       0.5474       0.6471       1.4837       0.8262
 11       0.5491       0.6428       1.4831       0.8292
 12       0.5492       0.6438       1.4798       0.8279
 13       0.5488       0.6457       1.4783       0.8263
```