NAG Toolbox

nag_inteq_volterra_weights (d05bw)

1 Purpose

nag_inteq_volterra_weights (d05bw) 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 Syntax

```
[omega, lensw, sw, ifail] = nag_inteq_volterra_weights(method, iorder, nomg, nwt)
[omega, lensw, sw, ifail] = d05bw(method, iorder, nomg, nwt)
```

3 Description

nag_inteq_volterra_weights (d05bw) 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:

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

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 9. 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 Parameters

5.1 Compulsory Input Parameters

1: **method** – CHARACTER(1)

The type of method to be used.

method = 'A'
For Adams' type formulae.

method = 'B'
For Backward Differentiation Formulae.

Constraint: **method** = 'A' or 'B'.

2: **iorder** – INTEGER

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

If **method** = 'A', $p =$ **iorder** $- 1$.

If **method** = 'B', $p =$ **iorder**.

Constraints:

if **method** = 'A', $3 \leq$ **iorder** ≤ 6;
if **method** = 'B', $2 \leq$ **iorder** ≤ 5.

3: **nomg** – INTEGER

The number of convolution weights, n_w.

Constraint: **nomg** ≥ 1.

4: **nwt** – INTEGER

p, the number of columns in the starting weights.

Constraints:

if **method** = 'A', **nwt** = **iorder** $- 1$;
if **method** = 'B', **nwt** = **iorder**.

5.2 Optional Input Parameters

None.

5.3 Output Parameters

1: **omega**(**nomg**) – REAL (KIND=nag_wp) array

Contains the first **nomg** convolution weights.

2: **lensw** – INTEGER

The number of rows in the weights $W_{i,j}$.

3: **sw**($ldsw$, **nwt**) – REAL (KIND=nag_wp) array

$ldsw = n$.

sw$(i, j+1)$ contains the weights $W_{i,j}$, for $i = 1, 2, \ldots,$ **lensw** and $j = 0, 1, \ldots,$ **nwt** $- 1$, where n is as defined in Section 3.

4: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

On entry, **method** $\neq$ 'A' or 'B'.

ifail $= 2$

On entry, **iorder** < 2 or **iorder** > 6,
or **nomg** < 1.

ifail $= 3$

On entry, **method** = 'A' and **iorder** $= 2$,
or **method** = 'B' and **iorder** $= 6$.

ifail $= 4$

On entry, **method** = 'A' and **nwt** $\neq$ **iorder** $- 1$,
or **method** = 'B' and **nwt** $\neq$ **iorder**.

ifail $= 5$

On entry, **method** = 'A' and $ldsw <$ **nomg** $+$ **iorder** $- 2$,
or **method** = 'B' and $ldsw <$ **nomg** $+$ **iorder** $- 1$.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

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

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

using nag_inteq_volterra_weights (d05bw). 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 (d05bw):

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

$$y_i = f(i \times h) + h\sum_{j=0}^{\mathbf{nwt}-1} \mathbf{sw}(i, j+1)K(i,h,j,h)y_j, \quad i = 1, 2, \ldots, \mathbf{nwt} - 1.$$

(b) Compute the inhomogeneous terms

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

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

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

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

```
function d05bw_example

fprintf('d05bw example results\n\n');

method = 'BDF';
iorder = nag_int(4);
nomg = nag_int(10);
nwt = nag_int(4);
[omega, lensw, sw, ifail] = d05bw( ...
    method, iorder, nomg, nwt);

fprintf('\nThe convolution weights\n\n');
n = [0:double(nomg)-1]';
fprintf('%3d    %10.4f\n',[n omega]');

fprintf('\nThe weights W\n\n');
n = [1:double(lensw)]';
fprintf('%3d    %10.4f%10.4f%10.4f%10.4f\n',[n sw]');
```

9.2 Program Results

```
d05bw example results

The convolution weights

  0        0.4800
  1        0.9216
  2        1.0783
  3        1.0504
  4        0.9962
  5        0.9797
  6        0.9894
  7        1.0003
  8        1.0034
  9        1.0017

The weights W

  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
```