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

```c
#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 $\omega_i$ 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:

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

(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 $W_{i,j}$, $\omega_i$ 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


5 Arguments

1:  `method` – Nag_ODEMethod

   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`.

Mark 25 d05bwc.1
2:  iorder – Integer  
   Input

   On entry: the order of the method to be used. The number of starting weights, \( p \) is determined by \texttt{method} and \texttt{iorder}.

   If \texttt{method} = Nag\_Adams, \( p = iorder - 1 \).

   If \texttt{method} = Nag\_BDF, \( p = iorder \).

   Constraints:
   \[ \begin{align*}
   & \text{if } \texttt{method} = \text{Nag\_Adams}, \ 3 \leq iorder \leq 6; \\
   & \text{if } \texttt{method} = \text{Nag\_BDF}, \ 2 \leq iorder \leq 5.
   \end{align*} \]

3:  nomg – Integer  
   Input

   On entry: the number of convolution weights, \( n_w \).

   Constraint: \( nomg \geq 1 \).

4:  omega[nomg] – double  
   Output

   On exit: contains the first \texttt{nomg} convolution weights.

5:  sw[n x p] – double  
   Output

   Note: the \((i, j)\)th element of the matrix is stored in \texttt{sw}[(j - 1) x n + i - 1].

   On exit: \texttt{sw}[(j x 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

\textbf{NE\_ALLOC\_FAIL}

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE\_BAD\_PARAM}

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

\textbf{NE\_ENUM\_INT}

On entry, \texttt{method} = Nag\_Adams and \texttt{iorder} = 2.
Constraint: if \texttt{method} = Nag\_Adams, \( 3 \leq iorder \leq 6 \).

On entry, \texttt{method} = Nag\_BDF and \texttt{iorder} = 6.
Constraint: if \texttt{method} = Nag\_BDF, \( 2 \leq iorder \leq 5 \).

On entry, \texttt{method} = \langle value \rangle and \texttt{iorder} = \langle value \rangle.
Constraint: if \texttt{method} = Nag\_Adams, \( 3 \leq iorder \leq 6 \).

On entry, \texttt{method} = \langle value \rangle and \texttt{iorder} = \langle value \rangle.
Constraint: if \texttt{method} = Nag\_BDF, \( 2 \leq iorder \leq 5 \).

\textbf{NE\_INT}

On entry, \texttt{iorder} = \langle value \rangle.
Constraint: \( 2 \leq iorder \leq 6 \).

On entry, \texttt{nomg} = \langle value \rangle.
Constraint: \( nomg \geq 1 \).
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

\[ y(t) = f(t) + \int_0^t K(t, s)y(s) \, ds, \quad 0 \leq t \leq T, \]  

(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

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

(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

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

(b) Compute the inhomogeneous terms

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

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

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

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).
10 Example

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

10.1 Program Text

/* nag_inteq_volterra_weights (d05bwc) Example Program. *
 * Copyright 2014 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*/
  #ifdef _WIN32
    scanf_s("%*[\n ]");
  #else
    scanf("%*[\n ]");
  #endif
  #ifdef _WIN32
    scanf_s("%9s%*[\n ]", methodstring, _countof(methodstring));
  #else
    scanf("%9s%*[\n ]", methodstring);
  #endif

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

  #ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[\n ]", &iorder);
  #else
    scanf("%"NAG_IFMT"%*[\n ]", &iorder);
  #endif
  #ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[\n ]", &nomg);
  #else
    scanf("%"NAG_IFMT"%*[\n ]", &nomg);
  #endif

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

  d05bwc.4

  NAG Library Manual
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("The convolution weights\n n-j omega\n");
for (j = 0; j < nomg; j++)
    printf("%3"NAG_IFMT" %10.4f\n", j+1, omega[j]);

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

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

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

#undef SW

END:

NAG_FREE(sw);
NAG_FREE(omega);

return exit_status;
}

10.2 Program Data

nag_inteq_volterra_weights (d05bwc) Example Program Data

<table>
<thead>
<tr>
<th>Method</th>
<th>iorder</th>
<th>nomg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nag_BDF</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>
10.3 Program Results

The convolution weights

<table>
<thead>
<tr>
<th>n-j</th>
<th>( n-j \omega )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4800</td>
</tr>
<tr>
<td>2</td>
<td>0.9216</td>
</tr>
<tr>
<td>3</td>
<td>1.0783</td>
</tr>
<tr>
<td>4</td>
<td>1.0504</td>
</tr>
<tr>
<td>5</td>
<td>0.9962</td>
</tr>
<tr>
<td>6</td>
<td>0.9797</td>
</tr>
<tr>
<td>7</td>
<td>0.9894</td>
</tr>
<tr>
<td>8</td>
<td>1.0034</td>
</tr>
<tr>
<td>9</td>
<td>1.0017</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

The weights \( W \)

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