NAG C Library Function Document

nag_1d_quad_wt_alglog_1 (d01spc)

1 Purpose

nag_1d_quad_wt_alglog_1 (d01spc) is an adaptive integrator which calculates an approximation to the integral of a function \( g(x)w(x) \) over a finite interval \([a,b]\):

\[
I = \int_a^b g(x)w(x) \, dx
\]

where the weight function \( w \) has end-point singularities of algebraico-logarithmic type.

2 Specification

```c
#include <nag.h>
#include <nagd01.h>

void nag_1d_quad_wt_alglog_1 (double (*g)(double x, Nag_User *comm),
   double a, double b, double alfa, double beta,
   Nag_QuadWeight wt_func, double epsabs, double epsrel,
   Integer max_num_subint, double *result, double *abserr,
   NAG_QuadProgress *qp, Nag_User *comm, NagError *fail)
```

3 Description

This function is based upon the QUADPACK routine QAWSE (Piessens et al. (1983)) and integrates a function of the form \( g(x)w(x) \), where the weight function \( w(x) \) may have algebraico-logarithmic singularities at the end-points \( a \) and/or \( b \). The strategy is a modification of that in nag_1d_quad_osc_1 (d01skc). We start by bisecting the original interval and applying modified Clenshaw–Curtis integration of orders 12 and 24 to both halves. Clenshaw–Curtis integration is then used on all sub-intervals which have \( a \) or \( b \) as one of their end-points (Piessens et al. (1974)). On the other sub-intervals Gauss–Kronrod (7–15 point) integration is carried out.

A ‘global’ acceptance criterion (as defined by Malcolm and Simpson (1976)) is used. The local error estimation control is described by Piessens et al. (1983).

4 Parameters

1: \( g \) – function supplied by user

The function \( g \), supplied by the user, must return the value of the function \( g \) at a given point.

The specification of \( g \) is:

```c
double g(double x, Nag_User *comm)
```

1: \( x \) – double

On entry: the point at which the function \( g \) must be evaluated.
2: \textbf{comm} – Nag_User * \\
\hspace{1em} On entry/on exit: pointer to a structure of type Nag_User with the following member:
\hspace{1em} \textbf{p} – Pointer \hspace{3em} \textit{Input/Output}
\hspace{1em} On entry/on exit: the pointer \textbf{comm->p} should be cast to the required type, e.g.,  
\hspace{1em} struct user *s = (struct user *)\textbf{comm->p}, to obtain the original object’s address with appropriate type. (See the argument \textbf{comm} below.)

2: \textbf{a} – double \hspace{3em} \textit{Input}
\hspace{1em} On entry: the lower limit of integration, \(a\).

3: \textbf{b} – double \hspace{3em} \textit{Input}
\hspace{1em} On entry: the upper limit of integration, \(b\).
\hspace{1em} Constraint: \(b > a\).

4: \textbf{alfa} – double \hspace{3em} \textit{Input}
\hspace{1em} On entry: the parameter \(\alpha\) in the weight function.
\hspace{1em} Constraint: \(\textbf{alfa} > -1.0\).

5: \textbf{beta} – double \hspace{3em} \textit{Input}
\hspace{1em} On entry: the parameter \(\beta\) in the weight function.
\hspace{1em} Constraint: \(\textbf{beta} > -1.0\).

6: \textbf{wt_func} – Nag_QuadWeight \hspace{3em} \textit{Input}
\hspace{1em} On entry: indicates which weight function is to be used:
\hspace{1em} if \textbf{wt_func} = Nag_Alg, \(w(x) = (x-a)^{\alpha}(b-x)^{\beta}\);  
\hspace{1em} if \textbf{wt_func} = Nag_Alg_loga, \(w(x) = (x-a)^{\alpha}(b-x)^{\beta}\ln(x-a)\);  
\hspace{1em} if \textbf{wt_func} = Nag_Alg_logb, \(w(x) = (x-a)^{\alpha}(b-x)^{\beta}\ln(b-x)\);  
\hspace{1em} if \textbf{wt_func} = Nag_Alg_loga_logb, \(w(x) = (x-a)^{\alpha}(b-x)^{\beta}\ln(x-a)\ln(b-x)\).
\hspace{1em} Constraint: \textbf{wt_func} = Nag_Alg, Nag_Alg_loga, Nag_Alg_logb, or Nag_Alg_loga_logb.

7: \textbf{epsabs} – double \hspace{3em} \textit{Input}
\hspace{1em} On entry: the absolute accuracy required. If \textbf{epsabs} is negative, the absolute value is used. See Section 6.1.

8: \textbf{epsrel} – double \hspace{3em} \textit{Input}
\hspace{1em} On entry: the relative accuracy required. If \textbf{epsrel} is negative, the absolute value is used. See Section 6.1.

9: \textbf{max_num_subint} – Integer \hspace{3em} \textit{Input}
\hspace{1em} On entry: the upper bound on the number of sub-intervals into which the interval of integration may be divided by the function. The more difficult the integrand, the larger \textbf{max_num_subint} should be.
\hspace{1em} \textit{Suggested values:} a value in the range 200 to 500 is adequate for most problems.
\hspace{1em} Constraint: \textbf{max_num_subint} \geq 2.
result – double *

On exit: the approximation to the integral I.

abserr – double *

On exit: an estimate of the modulus of the absolute error, which should be an upper bound for \(|I - \text{result}|\).

qp – Nag_quadProgress *

Pointer to structure of type Nag_quadProgress with the following members:

num_subint – Integer

On exit: the actual number of sub-intervals used.

fun_count – Integer

On exit: the number of function evaluations performed by nag_1d_quad_wt_alglog_1.

sub_int_beg_pts – double *
sub_int_end_pts – double *
sub_int_result – double *
sub_int_error – double *

On exit: these pointers are allocated memory internally with max_num_subint elements. If an error exit other than NE_INT_ARG_LT, NE_BAD_PARAM, NE_REAL_ARG_LE, NE_2_REAL_ARG_LE or NE_ALLOC_FAIL occurs, these arrays will contain information which may be useful. For details, see Section 6.

Before a subsequent call to nag_1d_quad_wt_alglog_1 is made, or when the information contained in these arrays is no longer useful, the user should free the storage allocated by these pointers using the NAG macro NAG_FREE.

comm – Nag_User *

On entry/on exit: pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/on exit: the pointer p, of type Pointer, allows the user to communicate information to and from the user-defined function g(). An object of the required type should be declared by the user, e.g., a structure, and its address assigned to the pointer p by means of a cast to Pointer in the calling program, e.g., comm.p = (Pointer)&s. The type Pointer is void *.

fail – NagError *

The NAG error parameter (see the Essential Introduction).

Users are recommended to declare and initialise fail and set fail.print = TRUE for this function.

5 Error Indicators and Warnings

NE_INT_ARG_LT

On entry, max_num_subint must not be less than 2: max_num_subint = <value>.

NE_BAD_PARAM

On entry, parameter wt_func had an illegal value.
**NE_REAL_ARG_LE**

On entry, \texttt{alpha} must not be less than or equal to \(-1.0\): \texttt{alpha} = \texttt{<value>}.  
On entry, \texttt{beta} must not be less than or equal to \(-1.0\): \texttt{beta} = \texttt{<value>}.  

**NE_2_REAL_ARG_LE**

On entry, \texttt{b} = \texttt{<value>} while \texttt{a} = \texttt{<value>}. These parameters must satisfy \texttt{b} > \texttt{a}.  

**NE_ALLOC_FAIL**

Memory allocation failed.  

**NE_QUAD_MAX_SUBDIV**

The maximum number of subdivisions has been reached: \texttt{max_num_subint} = \texttt{<value>}.  
The maximum number of subdivisions has been reached without the accuracy requirements being achieved. Look at the integrand in order to determine the integration difficulties. If the position of a discontinuity or a singularity of algebraic-logarithmic type within the interval can be determined, the interval must be split up at this point and the integrator called on the sub-intervals. If necessary, another integrator, which is designed for handling the type of difficulty involved, must be used. Alternatively, consider relaxing the accuracy requirements specified by \texttt{epsabs} and \texttt{epsrel}, or increasing the value of \texttt{max_num_subint}.  

**NE_QUAD_ROUNDOFF_TOL**

Round-off error prevents the requested tolerance from being achieved: \texttt{epsabs} = \texttt{<value>},  
\texttt{epsrel} = \texttt{<value>}.  
The error may be underestimated. Consider relaxing the accuracy requirements specified by \texttt{epsabs} and \texttt{epsrel}.  

**NE_QUAD_BAD_SUBDIV**

Extremely bad integrand behaviour occurs around the sub-interval (\texttt{<value>}, \texttt{<value>}). The same advice applies as in the case of \texttt{NE_QUAD_MAX_SUBDIV}.  

6 Further Comments

The time taken by \texttt{nag_1d_quad_wt_alglog} depends on the integrand and the accuracy required.  

If the function fails with an error exit other than \texttt{NE_REAL_ARG_LT}, \texttt{NE_BAD_PARAM}, \texttt{NE_REAL_ARG_LE}, \texttt{NE_2_REAL_ARG_LE} or \texttt{NE_ALLOC_FAIL} then the user may wish to examine the contents of the structure \texttt{qp}. These contain the end-points of the sub-intervals used by \texttt{nag_1d_quad_wt_alglog} along with the integral contributions and error estimates over these sub-intervals.  

Specifically, for \(i = 1, 2, \ldots, n\), let \(r_i\) denote the approximation to the value of the integral over the sub-interval \([a_i, b_i]\) in the partition of \([a, b]\) and \(e_i\) be the corresponding absolute error estimate.  

Then, \(\int_{a_i}^{b_i} g(x)w(x) dx \simeq r_i\) and \(\text{result} = \sum_{i=1}^{n} r_i\).  
The value of \(n\) is returned in \texttt{num_subint}, and the values \(a_i\), \(b_i\), \(r_i\) and \(e_i\) are stored in the structure \texttt{qp} as  

\[
\begin{align*}
  a_i & = \text{sub_int_beg_pts}[i-1], \\
  b_i & = \text{sub_int_end_pts}[i-1], \\
  r_i & = \text{sub_int_result}[i-1] \text{ and } \\
  e_i & = \text{sub_int_error}[i-1].
\end{align*}
\]
6.1 Accuracy

The function cannot guarantee, but in practice usually achieves, the following accuracy:

\[ |I - \text{result}| \leq \text{tol} \]

where

\[ \text{tol} = \max\{|\text{epsabs}|, |\text{epsrel}| \times |I|\} \]

and \text{epsabs} and \text{epsrel} are user-specified absolute and relative error tolerances. Moreover it returns the quantity \text{aberr} which, in normal circumstances, satisfies

\[ |I - \text{result}| \leq \text{aberr} \leq \text{tol}. \]

6.2 References


Piessens R, Mertens I and Branders M (1974) Integration of functions having end-point singularities Angew. Inf. 16 65–68


7 See Also

nag_1d_quad_gen_1 (d01sjc)

8 Example

To compute

\[ \int_{0}^{1} \ln x \cos(10\pi x) \, dx \]

and

\[ \int_{0}^{1} \frac{\sin(10x)}{\sqrt{x(1-x)}} \, dx. \]

8.1 Program Text

/* nag_1d_quad_wt_alglog_1(d01spc) Example Program *
 * Mark 6 revised, 2000.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <math.h>
#include <nagd01.h>
#include <nagx01.h>

static double f_sin(double x, Nag_User *comm);
static double f_cos(double x, Nag_User *comm);
main()
{
    static double alfa[2] = {0.0, -0.5};
    static double beta[2] = {0.0, -0.5};
    Nag_QuadWeight wt_func;
    double a, b;
    double epsabs, abserr, epsrel, result;
    static NagError fail;
    Nag_QuadProgress qp;
    Integer max_num_subint;
    int numfunc;
    NAG_D01SPC_FUN g;
    static char *Nag_QuadWeight_array[] =
    { // Nag_Alg, Nag_Alg_loga, Nag_Alg_logb,Nag_Alg_loga_logb};
    Boolean success = TRUE;
    Nag_User comm;
    Integer wt_array_ind;

    Vprintf("d01spc Example Program Results\n");
    epsabs = 0.0;
    epsrel = 0.0001;
    a = 0.0;
    b = 1.0;
    max_num_subint = 200;
    for (numfunc=0; numfunc < 2; ++numfunc)
    {
        switch (numfunc)
        {
        case 0:
            g = f_cos;
            wt_func = Nag_Alg_loga;
            wt_array_ind = 1;
            break;
        case 1:
            g = f_sin;
            wt_func = Nag_Alg;
            wt_array_ind = 0;
        }

d01spc(g, a, b, alfa[numfunc], beta[numfunc],
    wt_func, epsabs, epsrel, max_num_subint,
    &result, &abserr, &qp, &comm, &fail);

    Vprintf("a  - lower limit of integration = %10.4f\n", a);
    Vprintf("b  - upper limit of integration = %10.4f\n", b);
    Vprintf("epsabs - absolute accuracy requested = %9.2e\n", epsabs);
    Vprintf("epsrel - relative accuracy requested = %9.2e\n", epsrel);
    Vprintf("alpha - parameter in the weight function = %10.4f\n",
        alfa[numfunc]);
    Vprintf("beta - parameter in the weight function = %10.4f\n",
        beta[numfunc]);
    Vprintf("wt_func - denotes weight function to be \nused = %s\n", Nag_QuadWeight_array[wt_array_ind]);
    if (fail.code != NE_NOERROR)
        Vprintf("%s\n", fail.message);
    if (fail.code != NE_INT_ARG_LT && fail.code != NE_BAD_PARAM &&
        fail.code != NE_REAL_ARG_LE && fail.code != NE_2_REAL_ARG_LE &&
        fail.code != NE_ALLOC_FAIL)
{  
    Vprintf("result - approximation to the integral = %9.5f\n", result);
    Vprintf("abserr - estimate of the absolute error = %9.2e\n", abserr);
    Vprintf("qp.fun_count - number of function evaluations = %4ld\n", 
        qp.fun_count);
    Vprintf("qp.num_subint - number of subintervals used = %4ld\n", 
        qp.num_subint);
    /* Free memory used by qp */
    NAG_FREE(qp.sub_int_beg_pts);
    NAG_FREE(qp.sub_int_end_pts);
    NAG_FREE(qp.sub_int_result);
    NAG_FREE(qp.sub_int_error);
}

else
    success = FALSE;
}
if (success)
    exit(EXIT_SUCCESS);
else
    exit(EXIT_FAILURE);
}

static double f_cos(double x, Nag_User *comm)
{
    double a;
    double pi;

    pi = X01AAC;
    a = pi*10.0;
    return cos(a*x);
}

static double f_sin(double x, Nag_User *comm)
{
    double omega;

    omega = 10.0;
    return sin(omega*x);
}

8.2 Program Data

None.

8.3 Program Results

d01spc Example Program Results
a - lower limit of integration = 0.0000
b - upper limit of integration = 1.0000
epsabs - absolute accuracy requested = 0.00e+00
epsrel - relative accuracy requested = 1.00e-04
alfa - parameter in the weight function = 0.0000
beta - parameter in the weight function = 0.0000
wt_func - denotes weight function to be used = Nag_Alg_loga
result - approximation to the integral = -0.04899
abserr - estimate of the absolute error = 1.14e-07
qp.fun_count - number of function evaluations = 110
qp.num_subint - number of subintervals used =  4

a     - lower limit of integration =  0.0000
b     - upper limit of integration =  1.0000
epsabs - absolute accuracy requested =  0.00e+00
epsrel - relative accuracy requested =  1.00e-04

alfa   - parameter in the weight function =  -0.5000
beta   - parameter in the weight function =  -0.5000
wt_func - denotes weight function to be used = Nag_Alg
result - approximation to the integral =  0.53502
abserr - estimate of the absolute errox =  1.94e-12
qp.fun_count - number of function evaluations =  50
qp.num_subint - number of subintervals used =  2