

## NAG 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

```
#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

nag\_1d\_quad\_wt\_alglog\_1 (d01spc) 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 References

Malcolm M A and Simpson R B (1976) Local versus global strategies for adaptive quadrature *ACM Trans. Math. Software* **1** 129–146

Piessens R, de Doncker–Kapenga E, Überhuber C and Kahaner D (1983) *QUADPACK, A Subroutine Package for Automatic Integration* Springer–Verlag

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

#### 5 Arguments

- 1: **g** – function, supplied by the user *External Function*  
**g** must return the value of the function  $g$  at a given point.

The specification of **g** is:

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

- |    |   |              |
|----|---|--------------|
| 1: | <b>x</b> – double<br><i>On entry:</i> the point at which the function $g$ must be evaluated.  | <i>Input</i> |
| 2: | <b>comm</b> – Nag_User *<br>Pointer to a structure of type Nag_User with the following member:<br><br><b>p</b> – Pointer<br><i>On entry/exit:</i> the pointer <b>comm</b> → <b>p</b> should be cast to the required type, e.g.,<br><code>struct user *s = (struct user *)comm → p</code> , to obtain the original object's address with appropriate type. (See the argument <b>comm</b> below.) |              |
- 2:   **a** – double *Input*  
*On entry:* the lower limit of integration,  $a$ .
- 3:   **b** – double *Input*  
*On entry:* the upper limit of integration,  $b$ .  
*Constraint:* **b** > **a**.
- 4:   **alfa** – double *Input*  
*On entry:* the argument  $\alpha$  in the weight function.  
*Constraint:* **alfa** > -1.0.
- 5:   **beta** – double *Input*  
*On entry:* the argument  $\beta$  in the weight function.  
*Constraint:* **beta** > -1.0.
- 6:   **wt\_func** – Nag\_QuadWeight *Input*  
*On entry:* indicates which weight function is to be used:  
     if **wt\_func** = Nag\_Alg,  $w(x) = (x - a)^\alpha (b - x)^\beta$ ;  
     if **wt\_func** = Nag\_Alg\_loga,  $w(x) = (x - a)^\alpha (b - x)^\beta \ln(x - a)$ ;  
     if **wt\_func** = Nag\_Alg\_logb,  $w(x) = (x - a)^\alpha (b - x)^\beta \ln(b - x)$ ;  
     if **wt\_func** = Nag\_Alg\_loga\_logb,  $w(x) = (x - a)^\alpha (b - x)^\beta \ln(x - a) \ln(b - x)$ .  
*Constraint:* **wt\_func** = Nag\_Alg, Nag\_Alg\_loga, Nag\_Alg\_logb or Nag\_Alg\_loga\_logb.
- 7:   **epsabs** – double *Input*  
*On entry:* the absolute accuracy required. If **epsabs** is negative, the absolute value is used. See Section 7.
- 8:   **epsrel** – double *Input*  
*On entry:* the relative accuracy required. If **epsrel** is negative, the absolute value is used. See Section 7.
- 9:   **max\_num\_subint** – Integer *Input*  
*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 **max\_num\_subint** should be.  
*Constraint:* **max\_num\_subint** ≥ 2.

- 10: **result** – double \* *Output*  
*On exit:* the approximation to the integral  $I$ .
- 11: **abserr** – double \* *Output*  
*On exit:* an estimate of the modulus of the absolute error, which should be an upper bound for  $|I - \mathbf{result}|$ .
- 12: **qp** – Nag\_QuadProgress \*  
 Pointer to structure of type Nag\_QuadProgress with the following members:
- num\_subint** – Integer *Output*  
*On exit:* the actual number of sub-intervals used.
- fun\_count** – Integer *Output*  
*On exit:* the number of function evaluations performed by nag\_1d\_quad\_wt\_alglog\_1 (d01spc).
- sub\_int\_beg\_pts** – double \* *Output*  
**sub\_int\_end\_pts** – double \* *Output*  
**sub\_int\_result** – double \* *Output*  
**sub\_int\_error** – double \* *Output*
- 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 9.
- Before a subsequent call to nag\_1d\_quad\_wt\_alglog\_1 (d01spc) is made, or when the information contained in these arrays is no longer useful, you should free the storage allocated by these pointers using the NAG macro NAG\_FREE.
- 13: **comm** – Nag\_User \*  
 Pointer to a structure of type Nag\_User with the following member:
- p** – Pointer  
*On entry/exit:* the pointer **comm**→**p**, of type Pointer, allows you to communicate information to and from **g()**. An object of the required type should be declared, e.g., a structure, and its address assigned to the pointer **comm**→**p** by means of a cast to Pointer in the calling program, e.g., **comm.p** = (Pointer)&**s**. The type Pointer is void \*.
- 14: **fail** – NagError \* *Input/Output*  
 The NAG error argument (see Section 3.6 in the Essential Introduction).

## 6 Error Indicators and Warnings

### NE\_2\_REAL\_ARG\_LE

On entry, **b** =  $\langle \text{value} \rangle$  while **a** =  $\langle \text{value} \rangle$ . These arguments must satisfy **b** > **a**.

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

### NE\_BAD\_PARAM

On entry, argument **wt\_func** had an illegal value.

**NE\_INT\_ARG\_LT**

On entry, **max\_num\_subint** must not be less than 2: **max\_num\_subint** =  $\langle value \rangle$ .

**NE\_QUAD\_BAD\_SUBDIV**

Extremely bad integrand behaviour occurs around the sub-interval  $(\langle value \rangle, \langle value \rangle)$ .  
The same advice applies as in the case of NE\_QUAD\_MAX\_SUBDIV.

**NE\_QUAD\_MAX\_SUBDIV**

The maximum number of subdivisions has been reached: **max\_num\_subint** =  $\langle value \rangle$ .

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 algebraico-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 **epsabs** and **epsrel**, or increasing the value of **max\_num\_subint**.

**NE\_QUAD\_ROUNDOff\_TOL**

Round-off error prevents the requested tolerance from being achieved: **epsabs** =  $\langle value \rangle$ ,  
**epsrel** =  $\langle value \rangle$ .

The error may be underestimated. Consider relaxing the accuracy requirements specified by **epsabs** and **epsrel**.

**NE\_REAL\_ARG\_LE**

On entry, **alfa** =  $\langle value \rangle$ .

Constraint: **alfa**  $\leq -1.0$ .

On entry, **beta** =  $\langle value \rangle$ .

Constraint: **beta**  $\leq -1.0$ .

**7 Accuracy**

nag\_1d\_quad\_wt\_alglog\_1 (d01spc) 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 **epsabs** and **epsrel** are user-specified absolute and relative error tolerances. Moreover it returns the quantity **abserr** which, in normal circumstances, satisfies

$$|I - \text{result}| \leq \text{abserr} \leq \text{tol}.$$

**8 Parallelism and Performance**

Not applicable.

**9 Further Comments**

The time taken by nag\_1d\_quad\_wt\_alglog\_1 (d01spc) depends on the integrand and the accuracy required.

If the function fails with 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 then you may wish to examine the contents of the structure **qp**. These contain the end-points of the sub-intervals used by nag\_1d\_quad\_wt\_alglog\_1 (d01spc) along with the integral contributions and error estimates over these sub-intervals.

Specifically,  $i = 1, 2, \dots, n$ , 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 **result** =  $\sum_{i=1}^n r_i$ .

The value of  $n$  is returned in **qp**→**num\_subint**, and the values  $a_i$ ,  $b_i$ ,  $r_i$  and  $e_i$  are stored in the structure **qp** as

```

a_i = qp→sub_int_beg_pts[i - 1],
b_i = qp→sub_int_end_pts[i - 1],
r_i = qp→sub_int_result[i - 1] and
e_i = qp→sub_int_error[i - 1].

```

## 10 Example

This example computes

$$\int_0^1 \ln x \cos(10\pi x) dx$$

and

$$\int_0^1 \frac{\sin(10x)}{\sqrt{x(1-x)}} dx.$$

### 10.1 Program Text

```

/* nag_ld_quad_wt_alglog_1 (d01spc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 * Mark 6 revised, 2000.
 * Mark 7 revised, 2001.
 */

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

#ifdef __cplusplus
extern "C" {
#endif
static double NAG_CALL f_sin(double x, Nag_User *comm);
static double NAG_CALL f_cos(double x, Nag_User *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{
    /* Scalars */
    Integer          exit_status = 0;
    Integer          max_num_subint, wt_array_ind;
    int              numfunc;
    double           a, b, epsabs, abserr, epsrel, result;
    /* Arrays */
    static Integer use_comm[2] = {1, 1};
    static double  alpha[2] = { 0.0, -0.5 };
    static double  beta[2] = { 0.0, -0.5 };
    static const char *Nag_QuadWeight_array[] = { "Nag Alg", "Nag Alg_loga",

```

```

                                "Nag_Alq_logb", "Nag_Alq_loga_logb" };
/* Nag Types */
Nag_QuadProgress  qp;
Nag_QuadWeight   wt_func;
Nag_User         comm;
Nag_Error        fail;

INIT_FAIL(fail);

printf("nag_ld_quad_wt_alglog_1 (d01spc) Example Program Results\n");

/* For communication with user-supplied functions: */
comm.p = (Pointer)&use_comm;

epsabs = 0.0;
epsrel = 0.0001;
a = 0.0;
b = 1.0;
max_num_subint = 200;
for (numfunc = 0; numfunc < 2; ++numfunc)
{
    switch (numfunc)
    {
    default:
    case 0:
        wt_func = Nag_Alq_loga;
        wt_array_ind = 1;
        /* nag_ld_quad_wt_alglog_1 (d01spc).
         * One-dimensional adaptive quadrature, weight function with
         * end-point singularities of algebraic-logarithmic type,
         * thread-safe
         */
        nag_ld_quad_wt_alglog_1(f_cos, a, b, alpha[numfunc], beta[numfunc],
                                wt_func, epsabs, epsrel, max_num_subint,
                                &result, &abserr, &qp, &comm, &fail);
        printf("\nIntegral of cos(10*pi*x) on [a,b]\n");
        break;
    case 1:
        wt_func = Nag_Alq;
        wt_array_ind = 0;
        nag_ld_quad_wt_alglog_1(f_sin, a, b, alpha[numfunc], beta[numfunc],
                                wt_func, epsabs, epsrel, max_num_subint,
                                &result, &abserr, &qp, &comm, &fail);
        printf("\nIntegral of sin(10*x) on [a,b]\n");
    }
    printf("-----\n");
    printf("a          - lower limit of integration      = %9.4f\n", a);
    printf("b          - upper limit of integration          = %9.4f\n", b);
    printf("epsabs    - absolute accuracy requested          = %11.2e\n", epsabs);
    printf("epsrel    - relative accuracy requested          = %11.2e\n\n", epsrel);
    printf("alpha     - weight function parameter            = %9.4f\n",
        alpha[numfunc]);
    printf("beta     - weight function parameter            = %9.4f\n",
        beta[numfunc]);
    printf("wt_func   - weight function used                  = %s\n",
        Nag_QuadWeight_array[wt_array_ind]);

    if (fail.code != NE_NOERROR) printf("%s\n", fail.message);

    if (fail.code == NE_NOERROR || fail.code == NE_QUAD_BAD_SUBDIV ||
        fail.code == NE_QUAD_MAX_SUBDIV || fail.code == NE_QUAD_ROUNDOff_TOL)
    {
        printf("result    - approximation to the integral = %10.5f\n",
            result);
        printf("abserr   - estimate of absolute error      = %11.2e\n",
            abserr);
        printf("qp.fun_count - function evaluations        = %4\"NAG_IFMT\"\n",
            qp.fun_count);
        printf("qp.num_subint - subintervals used          = %4\"NAG_IFMT\"\n\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
    {
        exit_status = 1;
        goto END;
    }
}

END:
return exit_status;
}

static double NAG_CALL f_cos(double x, Nag_User *comm)
{
    double a;
    double pi;
    Integer *use_comm = (Integer *)comm->p;

    if (use_comm[0])
    {
        printf("(User-supplied callback f_cos, first invocation.)\n");
        use_comm[0] = 0;
    }

    /* nag_pi (x01aac). */
    pi = nag_pi;
    a = pi*10.0;
    return cos(a*x);
}

static double NAG_CALL f_sin(double x, Nag_User *comm)
{
    double omega;
    Integer *use_comm = (Integer *)comm->p;

    if (use_comm[1])
    {
        printf("(User-supplied callback f_sin, first invocation.)\n");
        use_comm[1] = 0;
    }

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

```

## 10.2 Program Data

None.

## 10.3 Program Results

nag\_ld\_quad\_wt\_alglog\_1 (d01spc) Example Program Results  
(User-supplied callback f\_cos, first invocation.)

Integral of  $\cos(10\pi x)$  on  $[a,b]$

```

-----
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

alpha  - weight function parameter    = 0.0000
beta   - weight function parameter    = 0.0000
wt_func - weight function used        = Nag_Algo_log
result - approximation to the integral = -0.04899

```

```
abserr - estimate of absolute error = 1.14e-07
qp.fun_count - function evaluations = 110
qp.num_subint - subintervals used = 4
```

(User-supplied callback f\_sin, first invocation.)

Integral of sin(10\*x) on [a,b]

```
-----
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

alpha  - weight function parameter = -0.5000
beta   - weight function parameter = -0.5000
wt_func - weight function used      = Nag_Alq
result - approximation to the integral = 0.53502
abserr - estimate of absolute error = 1.94e-12
qp.fun_count - function evaluations = 50
qp.num_subint - subintervals used = 2
```

---