

# NAG Library Function Document

## nag\_1d\_quad\_osc\_1 (d01skc)

### 1 Purpose

nag\_1d\_quad\_osc\_1 (d01skc) is an adaptive integrator, especially suited to oscillating, nonsingular integrands, which calculates an approximation to the integral of a function  $f(x)$  over a finite interval  $[a, b]$ :

$$I = \int_a^b f(x) dx.$$

### 2 Specification

```
#include <nag.h>
#include <nagd01.h>
void nag_1d_quad_osc_1 (
    double (*f)(double x, Nag_User *comm),
    double a, double b, 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\_osc\_1 (d01skc) is based upon the QUADPACK routine QAG (Piessens *et al.* (1983)). It is an adaptive function, using the Gauss 30-point and Kronrod 61-point rules. A ‘global’ acceptance criterion (as defined by Malcolm and Simpson (1976)) is used. The local error estimation is described by Piessens *et al.* (1983).

As this function is based on integration rules of high order, it is especially suitable for nonsingular oscillating integrands.

This function requires you to supply a function to evaluate the integrand at a single point.

### 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 (1973) An algorithm for automatic integration *Angew. Inf.* **15** 399–401

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

### 5 Arguments

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

The specification of **f** is:

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

1: **x** – double

*Input*

*On entry:* the point at which the integrand  $f$  must be evaluated.

- |    |   |  |
|----|---|--|
| 2: | <p><b>comm</b> – Nag_User *</p> <p>Pointer to a structure of type Nag_User with the following member:</p> <p><b>p</b> – Pointer</p> <p style="padding-left: 20px;"><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.)</p> |  |
|----|---|--|
- 2: **a** – double *Input*  
*On entry:* the lower limit of integration, *a*.
- 3: **b** – double *Input*  
*On entry:* the upper limit of integration, *b*. It is not necessary that  $a < b$ .
- 4: **epsabs** – double *Input*  
*On entry:* the absolute accuracy required. If **epsabs** is negative, the absolute value is used. See Section 7.
- 5: **epsrel** – double *Input*  
*On entry:* the relative accuracy required. If **epsrel** is negative, the absolute value is used. See Section 7.
- 6: **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**  $\geq 1$ .
- 7: **result** – double \* *Output*  
*On exit:* the approximation to the integral *I*.
- 8: **abserr** – double \* *Output*  
*On exit:* an estimate of the modulus of the absolute error, which should be an upper bound for  $|I - \mathbf{result}|$ .
- 9: **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\_osc\_1 (d01skc).
- 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 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_osc_1` (d01skc) 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`.

10: **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 **f()**. 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 *`.

11: **fail** – NagError \*

*Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

### NE\_INT\_ARG\_LT

On entry, **max\_num\_subint** must not be less than 1: **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 local difficulty within the interval can be determined (e.g., a singularity of the integrand or its derivative, a peak, a discontinuity, etc.) you will probably gain from splitting up the interval at this point and calling the integrator 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**.

## 7 Accuracy

`nag_1d_quad_osc_1` (d01skc) cannot guarantee, but in practice usually achieves, the following accuracy:

$$|I - \mathbf{result}| \leq tol$$

where

$$tol = \max\{|\mathbf{epsabs}|, |\mathbf{epsrel}| \times |I|\}$$

and **epsabs** and **epsrel** are user-specified absolute and relative error tolerances. Moreover it returns the quantity **abserr** which, in normal circumstances, satisfies

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

## 8 Parallelism and Performance

nag\_1d\_quad\_osc\_1 (d01skc) is not threaded in any implementation.

## 9 Further Comments

The time taken by nag\_1d\_quad\_osc\_1 (d01skc) depends on the integrand and the accuracy required.

If the function fails with an error exit other than NE\_INT\_ARG\_LT 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\_osc\_1 (d01skc) 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,  $\int_{a_i}^{b_i} f(x) dx \simeq r_i$  and  $\mathbf{result} = \sum_{i=1}^n r_i$ . 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

$$\begin{aligned} a_i &= \mathbf{qp} \rightarrow \mathbf{sub\_int\_beg\_pts}[i - 1], \\ b_i &= \mathbf{qp} \rightarrow \mathbf{sub\_int\_end\_pts}[i - 1], \\ r_i &= \mathbf{qp} \rightarrow \mathbf{sub\_int\_result}[i - 1] \text{ and} \\ e_i &= \mathbf{qp} \rightarrow \mathbf{sub\_int\_error}[i - 1]. \end{aligned}$$

## 10 Example

This example computes

$$\int_0^{2\pi} \sin(30x) \cos x dx.$$

### 10.1 Program Text

```

/* nag_1d_quad_osc_1 (d01skc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 *
 */

#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(double x, Nag_User *comm);

```

```

#ifdef __cplusplus
}
#endif

int main(void)
{
    static Integer use_comm[1] = { 1 };
    Integer exit_status = 0;
    double a, b;
    double epsabs, abserr, epsrel, result;
    Nag_QuadProgress qp;
    Integer max_num_subint;
    NagError fail;
    /* nag_pi (x0laac).
     * pi
     */
    double pi = nag_pi;
    Nag_User comm;

    INIT_FAIL(fail);

    printf("nag_ld_quad_osc_1 (d01skc) Example Program Results\n");

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

    epsabs = 0.0;
    epsrel = 0.001;
    a = 0.0;
    b = pi * 2.0;
    max_num_subint = 200;

    /* nag_ld_quad_osc_1 (d01skc).
     * One-dimensional adaptive quadrature, suitable for
     * oscillating functions, thread-safe
     */
    nag_ld_quad_osc_1(f, a, b, epsabs, epsrel, max_num_subint, &result, &abserr,
                     &qp, &comm, &fail);
    printf("a      - lower limit of integration = %10.4f\n", a);
    printf("b      - upper limit of integration = %10.4f\n", b);
    printf("epsabs - absolute accuracy requested = %11.2e\n", epsabs);
    printf("epsrel - relative accuracy requested = %11.2e\n\n", epsrel);
    if (fail.code != NE_NOERROR)
        printf("Error from nag_ld_quad_osc_1 (d01skc) %s\n", fail.message);
    if (fail.code != NE_INT_ARG_LT && fail.code != NE_ALLOC_FAIL &&
        fail.code != NE_NO_LICENCE) {
        printf("result - approximation to the integral = %9.5f\n", result);
        printf("abserr - estimate of the absolute error = %11.2e\n", abserr);
        printf("qp.fun_count - number of function evaluations = %4" NAG_IFMT
              "\n", qp.fun_count);
        printf("qp.num_subint - number of subintervals used = %4" NAG_IFMT "\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(double x, Nag_User *comm)
{
    Integer *use_comm = (Integer *) comm->p;

```

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

return x * sin(x * 30.0) * cos(x);
}
```

## 10.2 Program Data

None.

## 10.3 Program Results

```
nag_1d_quad_osc_1 (d01skc) Example Program Results
(User-supplied callback f, first invocation.)
a      - lower limit of integration =    0.0000
b      - upper limit of integration =    6.2832
epsabs - absolute accuracy requested =    0.00e+00
epsrel - relative accuracy requested =    1.00e-03

result - approximation to the integral = -0.20967
abserr - estimate of the absolute error =  4.48e-14
qp.fun_count - number of function evaluations = 427
qp.num_subint - number of subintervals used = 4
```

---