NAG C 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, non-singular 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

This function 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 non-singular oscillating integrands.

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

4 Parameters

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

The function \( f \), supplied by the user, 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

On entry: the point at which the integrand \( f \) must be evaluated.

2: \( \text{comm} \) – Nag_User *

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

\( p \) – Pointer

On entry/exit: the pointer \( \text{comm} \rightarrow p \) should be cast to the required type, e.g.,

\[
\text{struct user } *s = (\text{struct user } *)\text{comm} \rightarrow p;
\]

To obtain the original object’s address with appropriate type. (See the argument \( \text{comm} \) below.)

2: \( a \) – double

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: \( \text{epsabs} \) – double

*Input*

*On entry*: the absolute accuracy required. If \( \text{epsabs} \) is negative, the absolute value is used. See Section 6.1.

5: \( \text{epsrel} \) – double

*Input*

*On entry*: the relative accuracy required. If \( \text{epsrel} \) is negative, the absolute value is used. See Section 6.1.

6: \( \text{max_num_subint} \) – 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 \( \text{max_num_subint} \) should be.

*Suggested values*: a value in the range 200 to 500 is adequate for most problems.

*Constraint*: \( \text{max_num_subint} \geq 1 \).

7: \( \text{result} \) – double *

*Output*

*On exit*: the approximation to the integral \( I \).

8: \( \text{abser}r \) – double *

*Output*

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

9: \( \text{qp} \) – Nag_QuadProgress *

Pointer to structure of type Nag_QuadProgress with the following members:

\[
\begin{align*}
\text{num_subint} & \quad \text{Integer} & \quad \text{Output} \\
\text{fun_count} & \quad \text{Integer} & \quad \text{Output} \\
\text{sub_int_beg_pts} & \quad \text{double *} & \quad \text{Output} \\
\text{sub_int_end_pts} & \quad \text{double *} & \quad \text{Output} \\
\text{sub_int_result} & \quad \text{double *} & \quad \text{Output} \\
\text{sub_int_error} & \quad \text{double *} & \quad \text{Output}
\end{align*}
\]

*On exit*: these pointers are allocated memory internally with \( \text{max_num_subint} \) elements. If an error exit other than \text{NE_INT_ARG_LT} or \text{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_osc_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 \text{NAG_FREE}. 

\[d01skc.2\]
10:  \texttt{comm} – Nag_User *  

\textit{On entry/on exit:} pointer to a structure of type \texttt{Nag_User} with the following member:  

\texttt{p} – Pointer \hspace{1cm} \textit{Input/Output}

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

11:  \texttt{fail} – NagError * \hspace{1cm} \textit{Input/Output}

The NAG error parameter (see the Essential Introduction).

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

5 \quad \textbf{Error Indicators and Warnings}

\textbf{NE_INT_ARG_LT}  

On entry, \texttt{max_num_subint} must not be less than 1: \texttt{max_num_subint} = <value>.

\textbf{NE_ALLOCS_FAIL}  

Memory allocation failed.

\textbf{NE_QUAD_MAX_SUBDIV}  

The maximum number of subdivisions has been reached: \texttt{max_num_subint} = <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 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 \texttt{epsabs} and \texttt{epsrel}, or increasing the value of \texttt{max_num_subint}.

\textbf{NE_QUAD_ROUNDING_TOL}  

Round-off error prevents the requested tolerance from being achieved: \texttt{epsabs} = <value>, \texttt{epsrel} = <value>.  

The error may be underestimated. Consider relaxing the accuracy requirements specified by \texttt{epsabs} and \texttt{epsrel}.

\textbf{NE_QUAD_BAD_SUBDIV}  

Extremely bad integrand behaviour occurs around the sub-interval (<value>, <value>).  

The same advice applies as in the case of \textbf{NE_QUAD_MAX_SUBDIV}.

6 \quad \textbf{Further Comments}

The time taken by \texttt{nag_1d_quad_osc_1} depends on the integrand and the accuracy required.

If the function fails with an error exit other than \textbf{NE_INT_ARG_LT} or \textbf{NE_ALLOCS_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_osc_1} 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, $\int_{a_i}^{b_i} f(x) dx \simeq r_i$ and `result = \sum_{i=1}^{n} r_i`. The value of `n` is returned in `num_subint`, and the values $a_i$, $b_i$, $r_i$ and $e_i$ are stored in the structure `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:

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

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 `abser` which, in normal circumstances, satisfies

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

### 6.2 References


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

### 7 See Also

nag_1d_quad_gen_1 (d01sjc)
nag_1d_quad_brkpts_1 (d01slc)

### 8 Example

To compute

$$
\int_{0}^{2\pi} \sin(30x) \cos x \, dx.
$$

### 8.1 Program Text

```c
#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
```

---

*d01skc:4 [NP3491/6]*
#include <math.h>
#include <nagd01.h>
#include <nagx01.h>

static double f(double x, Nag_User *comm);

main()
{
    double a, b;
    double epsabs, abseps, epsrel, result;
    Nag_QuadProgress qp;
    Integer max_num_subint;
    static NagError fail;
    double pi = X01AAC;
    Nag_User comm;

    Vprintf("d01skc Example Program Results\n");
    epsabs = 0.0;
    epsrel = 0.001;
    a = 0.0;
    b = pi * 2.0;
    max_num_subint = 200;

d01skc(f, a, b, epsabs, abseps, epsrel, max_num_subint, &result, &abseps, &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\n", epsrel);
    if (fail.code != NE_NOERROR)
        Vprintf("%s\n", fail.message);
    if (fail.code != NE_INT_ARG_LT & fail.code != NE_ALLOC_FAIL)
    {
        Vprintf("result - approximation to the integral = %9.5f\n", result);
        Vprintf("abseps - estimate of the absolute error = %9.2e\n", abseps);
        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);
        exit(EXIT_SUCCESS);
    }
    else
        exit(EXIT_FAILURE);
}

static double f(double x, Nag_User *comm)
{
    return x*sin(x*30.0)*cos(x);
}

8.2 Program Data

None.
8.3 Program Results

d01skc Example Program Results
\[ \begin{align*}
 a & \quad \text{lower limit of integration} = \quad 0.0000 \\
 b & \quad \text{upper limit of integration} = \quad 6.2832 \\
 \text{epsabs} & \quad \text{absolute accuracy requested} = \quad 0.00e+00 \\
 \text{epsrel} & \quad \text{relative accuracy requested} = \quad 1.00e-03 \\
 \text{result} & \quad \text{approximation to the integral} = \quad -0.20967 \\
 \text{abserr} & \quad \text{estimate of the absolute error} = \quad 4.49e-14 \\
 \text{qp.fun_count} & \quad \text{number of function evaluations} = \quad 427 \\
 \text{qp.num_subint} & \quad \text{number of subintervals used} = \quad 4 \\
\end{align*} \]