nag_1d_quad_inf_wt_trig_1 (d01ssc)

1. Purpose

nag_1d_quad_inf_wt_trig_1 (d01ssc) calculates an approximation to the sine or the cosine transform of a function g over $[a, \infty)$:

$$I = \int_a^\infty g(x)\sin(\omega x)\,dx \quad \text{or} \quad I = \int_a^\infty g(x)\cos(\omega x)\,dx$$

(for a user-specified value of ω).

2. Specification

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

void nag_1d_quad_inf_wt_trig_1(double (*g)(double x, Nag_User *comm),
               double a, double omega, Nag_TrigTransform wt_func,
               Integer maxintervals, Integer maxsubints_per_interval,
               double epsabs, double *result, double *abserr,
               Nag_QuadSubProgress *qpsub, Nag_User *comm, NagError *fail)
```

3. Description

This function is based upon the QUADPACK routine QAWFE (Piessens *et al.* (1983)). It is an adaptive routine, designed to integrate a function of the form $g(x)w(x)$ over a semi-infinite interval, where $w(x)$ is either $\sin(\omega x)$ or $\cos(\omega x)$. Over successive intervals

$$C_k = [a + (k-1) \times c, a + k \times c], \quad k = 1, 2, \ldots, \textbf{qpsub.intervals}$$

integration is performed by the same algorithm as is used by nag_1d_quad_wt_trig_1 (d01snc). The intervals C_k are of constant length

$$c = \{2[|\omega|] + 1\}\pi/|\omega|, \quad \omega \neq 0,$$

where $[|\omega|]$ represents the largest integer less than or equal to $|\omega|$. Since c equals an odd number of half periods, the integral contributions over succeeding intervals will alternate in sign when the function g is positive and monotonically decreasing over $[a, \infty)$. The algorithm, described by Piessens *et al.* (1983), incorporates a global acceptance criterion (as defined by Malcolm and Simpson (1976)) together with the ϵ-algorithm (Wynn (1956)) to perform extrapolation. The local error estimation is described by Piessens *et al.* (1983).

If $\omega = 0$ and **wt_func** = **Nag_Cosine**, the routine uses the same algorithm as nag_1d_quad_inf_1 (d01smc) (with **epsrel** = 0.0).

In contrast to most other functions in Chapter d01, this function works only with a user-specified absolute error tolerance (**epsabs**). Over the interval C_k it attempts to satisfy the absolute accuracy requirement

$$EPSA_k = U_k \times \textbf{epsabs},$$

where $U_k = (1-p)p^{k-1}$, for $k = 1, 2, \ldots$ and $p = 0.9$.

However, when difficulties occur during the integration over the kth interval C_k such that the error flag **qpsub->interval_flag[k−1]** is non-zero, the accuracy requirement over subsequent intervals is relaxed. See Piessens *et al.* (1983) for more details.

4. Parameters

g

The function **g**, supplied by the user, must return the value of the function g at a given point. The specification of **g** is:

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

x
Input: the point at which the function g must be evaluated.

comm
Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer
Input/Output: the pointer **comm->p** should be cast to the required type, e.g. `struct user *s = (struct user *)comm->p`, to obtain the original object's address with appropriate type. (See the argument **comm** below.)

a
Input: the lower limit of integration, a.

omega
Input: the parameter ω in the weight function of the transform.

wt_func
Input: indicates which integral is to be computed:
if **wt_func** = **Nag_Cosine**, $w(x) = \cos(\omega x)$;
if **wt_func** = **Nag_Sine**, $w(x) = \sin(\omega x)$;
Constraint: **wt_func** = **Nag_Cosine** or **Nag_Sine**.

maxintervals
Input: an upper bound on the number of intervals C_k needed for the integration.
Suggested value: **maxintervals** = 50 is adequate for most problems.
Constraint: **maxintervals** ≥ 3.

maxsubints_per_interval
Input: the upper bound on the number of sub-intervals into which any of the intervals, C_k, may be divided by the function.
Suggested value: a value in the range 200 to 500 is adequate for most problems.
Constraint: **maxsubints_per_interval** ≥ 1.

epsabs
Input: the absolute accuracy required. If **epsabs** is negative, the absolute value is used. See Section 6.1.

result
Output: the approximation to the integral I.

abserr
Output: an estimate of the modulus of the absolute error, which should be an upper bound for $|I - \textbf{result}|$.

qpsub
Pointer to structure of type Nag_QuadSubProgress with the following members:

intervals – Integer
Output: the number of intervals C_k actually used for the integration.

fun_count – Integer
Output: the number of function evaluations performed by this function.

subints_per_interval – Integer
Output: the maximum number of sub-intervals actually used for integrating over any of the intervals C_k.

interval_error – double *
Output: the error estimate corresponding to the integral contribution over the interval C_k, for $k = 1, 2, \ldots,$**qpsub.intervals**.

interval_result – double *
Output: the corresponding integral contribution over the interval C_k for $k = 1, 2, \ldots,$**qpsub.intervals**.

interval_flag – Integer *
Output: the error flag corresponding to **qpsub.interval_result**, for $k = 1, 2, \ldots,$**qpsub.intervals**. See also Section 5.

comm
Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer
Input/Output: 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 will be `void *` with a C compiler that defines `void *` and `char *` otherwise.

fail
The NAG error parameter, see the Essential Introduction to the NAG C Library.
Users are recommended to declare and initialize **fail** and set **fail.print** = TRUE for this function.

5. Error Indications and Warnings

NE_INT_ARG_LT
On entry, **maxsubints_per_interval** must not be less than 1: **maxsubints_per_interval** = ⟨*value*⟩.
On entry, **maxintervals** must not be less than 3: **maxintervals** = ⟨*value*⟩.

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

NE_ALLOC_FAIL
Memory allocation failed.

NE_QUAD_MAX_SUBDIV
The maximum number of subdivisions has been reached: **maxsubints_per_interval** = ⟨*value*⟩. The maximum number of subdivisions within an interval 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 this function on the infinite subrange and an appropriate integrator on the finite subrange. Alternatively, consider relaxing the accuracy requirements specified by **epsabs** or increasing the value of **maxsubints_per_interval**.

NE_QUAD_ROUNDOFF_ABS_TOL
Round-off error prevents the requested tolerance from being achieved: **epsabs** = ⟨*value*⟩.
The error may be underestimated. Consider relaxing the accuracy requirements specified by **epsabs**.

NE_QUAD_BAD_SUBDIV
Extremely bad integrand behaviour occurs around the sub-interval (⟨*value*⟩, ⟨*value*⟩).
The same advice applies as in the case of **NE_QUAD_MAX_SUBDIV**.

NE_QUAD_ROUNDOFF_EXTRAPL
Round-off error is detected during extrapolation.

The requested tolerance cannot be achieved, because the extrapolation does not increase the accuracy satisfactorily; the returned result is the best which can be obtained. The same advice applies as in the case of **NE_QUAD_MAX_SUBDIV**.

NE_QUAD_BAD_SUBDIV_INT
Bad integration behaviour has occured within one or more intervals.

NE_QUAD_MAX_INT
Maximum number of intervals allowed has been achieved. Increase the value of **maxintervals**.

NE_QUAD_EXTRAPL_INT
The extrapolation table constructed for convergence acceleration of the series formed by the integral contribution over the integral does not converge.

In the cases where **fail.code** = **NE_QUAD_BAD_SUBDIV_INT**, **NE_QUAD_MAX_INT** or **NE_QUAD_EXTRAPL_INT**, additional information about the cause of the error can be obtained from the array **qpsub->interval_flag**, as follows:

qpsub.interval_flag$[k-1]$ = **1**
The maximum number of subdivisions (= **maxsubints_per_interval**) has been achieved on the kth interval.

qpsub.interval_flag$[k-1]$ = **2**
Occurrence of round-off error is detected and prevents the tolerance imposed on the kth interval from being achieved.

qpsub.interval_flag$[k-1]$ = **3**
Extremely bad integrand behaviour occurs at some points of the kth interval.

qpsub.interval_flag$[k-1]$ = **4**
The integration procedure over the kth interval does not converge (to within the required accuracy) due to round-off in the extrapolation procedure invoked on this interval. It is assumed that the result on this interval is the best which can be obtained.

qpsub.interval_flag$[k-1]$ = **5**
The integral over the kth interval is probably divergent or slowly convergent. It must be noted that divergence can occur with any other value of **qpsub.interval_flag**$[k-1]$.

If users declare and initialise **fail** and set **fail.print** = **TRUE** as recommended then

NE_QUAD_NO_CONV
The integral is probably divergent or slowly convergent. Please note that divergence can occur with any error exit other than **NE_INT_ARG_LT**, **NE_BAD_PARAM** or **NE_ALLOC_FAIL**.

may be produced supplemented by messages indicating more precisely where problems were encountered by the function. However, if the default error handling, `NAGERR_DEFAULT`, is used then one of the following errors may occur. Please note the program will terminate when the first of such errors is detected.

NE_QUAD_MAX_SUBDIV_SPEC_INT
The maximum number of subdivisions has been reached,
maxsubints_per_interval = ⟨*value*⟩ on the ⟨*value*⟩ interval.
interval_flag[⟨*value*⟩] = ⟨*value*⟩ over sub-interval (⟨*value*⟩, ⟨*value*⟩).

NE_QUAD_ROUNDOFF_TOL_SPEC_INT
Round-off error prevents the requested tolerance from being achieved on the ⟨*value*⟩ interval.
interval_flag[⟨*value*⟩] = ⟨*value*⟩ over sub-interval (⟨*value*⟩, ⟨*value*⟩).

NE_QUAD_BAD_SPEC_INT
Bad integrand behaviour occurs at some points of the ⟨*value*⟩ interval.
interval_flag[⟨*value*⟩] = ⟨*value*⟩ over sub-interval (⟨*value*⟩, ⟨*value*⟩).

NE_QUAD_NO_CONV_SPEC_INT
The integral has failed to converge on the ⟨*value*⟩ interval.
interval_flag[⟨*value*⟩] = ⟨*value*⟩ over sub-interval (⟨*value*⟩, ⟨*value*⟩).

NE_QUAD_DIVERGENCE_SPEC_INT
The integral is probably divergent on the ⟨*value*⟩ interval.
interval_flag[⟨*value*⟩] = ⟨*value*⟩ over sub-interval (⟨*value*⟩, ⟨*value*⟩).

6. Further Comments

The time taken depends on the integrand and on the accuracy required.

6.1. Accuracy

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

$$|I - \textbf{result}| \leq |\textbf{epsabs}|.$$

where **epsabs** is the user-specified absolute error tolerance. Moreover it returns the quantity **abserr** which, in normal circumstances, satisfies

$$|I - \textbf{result}| \leq \textbf{abserr} \leq |\textbf{epsabs}|.$$

6.2. References

Malcolm M A and Simpson R B (1976) Local Versus Global Strategies for Adaptive Quadrature *ACM Trans. Math. Softw.* **1** 129–146.

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

Wynn P (1956) On a Device for Computing the $e_m(S_n)$ Transformation *Math. Tables Aids Comput.* **10** 91–96.

7. See Also

nag_1d_quad_inf_1 (d01smc)
nag_1d_quad_wt_trig_1 (d01snc)

8. Example

To compute

$$\int_0^\infty \frac{1}{\sqrt{x}} \cos(\pi x/2)\, dx.$$

8.1. Program Text

```
/* nag_1d_quad_inf_wt_trig_1(d01ssc) Example Program
 *
 * Copyright 1998 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 */

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

#ifdef NAG_PROTO
static double g(double x, Nag_User *comm);
#else
static double g();
#endif

main()
{

  double a;
  double omega;
  double epsabs, abserr;
  Nag_TrigTransform wt_func;
  double result;
```

```
  Nag_QuadSubProgress qpsub;
  Integer maxintervals, maxsubint_per_int;
  static NagError fail;
  Nag_User comm;

  Vprintf("d01ssc Example Program Results\n");
  epsabs = 0.001;
  a = 0.0;
  omega = X01AAC * 0.5;
  wt_func = Nag_Cosine;
  maxintervals = 50;
  maxsubint_per_int = 500;

  d01ssc(g, a, omega, wt_func, maxintervals, maxsubint_per_int,
         epsabs, &result, &abserr, &qpsub, &comm, &fail);

  Vprintf("a      - lower limit of integration = %10.4f\n", a);
  Vprintf("b      - upper limit of integration = infinity\n");
  Vprintf("epsabs - absolute accuracy requested = %9.2e\n\n", epsabs);
  if (fail.code != NE_NOERROR)
    Vprintf("%s\n", fail.message);
  if (fail.code != NE_INT_ARG_LT)
    {
      Vprintf("result - approximation to the integral = %9.5f\n", result);
      Vprintf("abserr - estimate of the absolute error = %9.2e\n", abserr);
      Vprintf("qpsub.fun_count  - number of function evaluations = %4ld\n",
              qpsub.fun_count);
      Vprintf("qpsub.intervals  - number of intervals used = %4ld\n",
              qpsub.intervals);
      Vprintf("qpsub.subints_per_interval  - \n\
      maximum number of subintervals used in any one interval = %4ld\n",
              qpsub.subints_per_interval);
      exit(EXIT_SUCCESS);
    }
  else
    exit(EXIT_FAILURE);
}


#ifdef NAG_PROTO
static double g(double x, Nag_User *comm)
#else
     static double g(x, comm)
     double x;
     Nag_User *comm;
#endif
{
  return (x > 0.0) ? 1.0/sqrt(x) : 0.0;
}
```

8.2. Program Data

None.

8.3. Program Results

```
d01ssc Example Program Results
a      - lower limit of integration =     0.0000
b      - upper limit of integration = infinity
epsabs - absolute accuracy requested =  1.00e-03

result - approximation to the integral =   1.00000
abserr - estimate of the absolute error =  5.92e-04
qpsub.fun_count  - number of function evaluations =  380
qpsub.intervals  - number of intervals used =    6
qpsub.subints_per_interval  -
      maximum number of subintervals used in any one interval =    8
```