NAG Library Function Document

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 max_num_subint, double epsabs, double *result, double *abserr,
     Nag_QuadSubProgress *qpsub, Nag_User *comm, NagError *fail)
```

3 Description

nag_1d_quad_inf_wt_trig_1 (d01ssc) is based upon the QUADPACK routine QAWFE (Piessens *et al.* (1983)). It is an adaptive function, 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

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

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 $\textbf{wt\_func} = \text{Nag\_Cosine}$, the function uses the same algorithm as nag_1d_quad_inf_1 (d01smc) (with $\textbf{epsrel} = 0.0$).

In contrast to most other functions in Chapter d01, nag_1d_quad_inf_wt_trig_1 (d01ssc) 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 $\textbf{qpsub}\rightarrow\textbf{interval\_flag}[k-1]$ is nonzero, the accuracy requirement over subsequent intervals is relaxed. See Piessens *et al.* (1983) for more details.

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

Wynn P (1956) On a device for computing the $e_m(S_n)$ transformation *Math. Tables Aids Comput.* **10** 91–96

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: **x** – double *Input*

On entry: the point at which the function g must be evaluated.

2: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: 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.)

2: **a** – double *Input*

On entry: the lower limit of integration, a.

3: **omega** – double *Input*

On entry: the argument ω in the weight function of the transform.

4: **wt_func** – Nag_TrigTransform *Input*

On entry: 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.

5: **maxintervals** – Integer *Input*

On entry: 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.

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** ≥ 1.

7: **epsabs** – double *Input*

On entry: the absolute accuracy required. If **epsabs** is negative, the absolute value is used. See Section 7.

8: **result** – double * *Output*

On exit: the approximation to the integral I.

9: **abserr** – double * *Output*

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

10: **qpsub** – Nag_QuadSubProgress *

Pointer to structure of type Nag_QuadSubProgress with the following members:

intervals – Integer *Output*

On exit: the number of intervals C_k actually used for the integration.

fun_count – Integer *Output*

On exit: the number of function evaluations performed by nag_1d_quad_inf_wt_trig_1 (d01ssc).

subints_per_interval – Integer * *Output*

On exit: the maximum number of sub-intervals actually used for integrating over any of the intervals C_k.

interval_error – double * *Output*

On exit: the error estimate corresponding to the integral contribution over the interval C_k, for $k = 1, 2, \ldots,$ **intervals**.

interval_result – double * *Output*

On exit: the corresponding integral contribution over the interval C_k, for $k = 1, 2, \ldots,$ **intervals**.

interval_flag – Integer * *Output*

On exit: the error flag corresponding to **interval_result**, for $k = 1, 2, \ldots,$ **intervals**. See also Section 6.

When the information available in the arrays **interval_error**, **interval_result** and **interval_flag** is no longer useful, or before a subsequent call to nag_1d_quad_inf_wt_trig_1 (d01ssc) with the same argument **qpsub** is made, you should free the storage contained in this pointer using the NAG macro `NAG_FREE`. Note that these arrays do not need to be freed if one of the error exits NE_INT_ARG_LT, NE_BAD_PARAM or NE_ALLOC_FAIL occurred.

11: **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 *`.

12: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

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 (= **max_num_subint**) 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 you declare and initialize **fail** and set **fail.print** = Nag_TRUE as recommended then NE_QUAD_NO_CONV 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 NE_QUAD_MAX_SUBDIV_SPEC_INT, NE_QUAD_ROUNDOFF_TOL_SPEC_INT, NE_QUAD_BAD_SPEC_INT, NE_QUAD_NO_CONV_SPEC_INT and NE_QUAD_DIVERGENCE_SPEC_INT may occur. Please note the program will terminate when the first of such errors is detected.

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, **maxintervals** = ⟨*value*⟩.
Constraint: **maxintervals** ≥ 3.

On entry, **max_num_subint** must not be less than 1: **max_num_subint** = ⟨*value*⟩.

NE_QUAD_BAD_SPEC_INT

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

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_BAD_SUBDIV_INT

Bad integration behaviour has occurred within one or more intervals.

NE_QUAD_DIVERGENCE_SPEC_INT

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

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.

NE_QUAD_MAX_INT

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

NE_QUAD_MAX_SUBDIV

The maximum number of subdivisions has been reached: **max_num_subint** = ⟨*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 **max_num_subint**.

NE_QUAD_MAX_SUBDIV_SPEC_INT

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

NE_QUAD_NO_CONV

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

NE_QUAD_NO_CONV_SPEC_INT

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

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_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 that can be obtained.
The same advice applies as in the case of NE_QUAD_MAX_SUBDIV.

NE_QUAD_ROUNDOFF_TOL_SPEC_INT

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

7 Accuracy

nag_1d_quad_inf_wt_trig_1 (d01ssc) cannot guarantee, but in practice usually achieves, the following accuracy:

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

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

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

8 Parallelism and Performance

Not applicable.

9 Further Comments

The time taken by nag_1d_quad_inf_wt_trig_1 (d01ssc) depends on the integrand and on the accuracy required.

10 Example

This example computes

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

10.1 Program Text

```
/* nag_1d_quad_inf_wt_trig_1 (d01ssc) 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 g(double x, Nag_User *comm);
#ifdef __cplusplus
```

```
}
#endif

int main(void)
{
  static Integer use_comm[1] = {1};
  Integer             exit_status = 0;
  double              a;
  double              omega;
  double              epsabs, abserr;
  Nag_TrigTransform   wt_func;
  double              result;
  Nag_QuadSubProgress qpsub;
  Integer             maxintervals, maxsubint_per_int;
  NagError            fail;
  Nag_User            comm;

  INIT_FAIL(fail);

  printf(
         "nag_1d_quad_inf_wt_trig_1 (d01ssc) Example Program Results\n");

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

  epsabs = 0.001;
  a = 0.0;
  /* nag_pi (x01aac).
   * pi
   */
  omega = nag_pi * 0.5;
  wt_func = Nag_Cosine;
  maxintervals = 50;
  maxsubint_per_int = 500;

  /* nag_1d_quad_inf_wt_trig_1 (d01ssc).
   * One-dimensional adaptive quadrature, semi-infinite
   * interval, sine or cosine weight function, thread-safe
   */
  nag_1d_quad_inf_wt_trig_1(g, a, omega, wt_func, maxintervals,
                            maxsubint_per_int, epsabs, &result, &abserr,
                            &qpsub,
                            &comm,
                            &fail);
  printf("a      - lower limit of integration = %10.4f\n", a);
  printf("b      - upper limit of integration = infinity\n");
  printf("epsabs - absolute accuracy requested = %11.2e\n\n", epsabs);
  if (fail.code != NE_NOERROR)
    printf("Error from nag_1d_quad_inf_wt_trig_1 (d01ssc) %s\n",
            fail.message);
  if (fail.code != NE_INT_ARG_LT && fail.code != NE_BAD_PARAM &&
      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("qpsub.fun_count  - number of function evaluations ="
              " %4"NAG_IFMT"\n", qpsub.fun_count);
      printf("qpsub.intervals  - number of intervals used = %4"NAG_IFMT"\n",
              qpsub.intervals);
      printf("qpsub.subints_per_interval  - \n"
              "maximum number of subintervals used in any one interval ="
              " %4"NAG_IFMT"\n", qpsub.subints_per_interval);
      /* Free memory used by qpsub */
      NAG_FREE(qpsub.interval_error);
      NAG_FREE(qpsub.interval_result);
      NAG_FREE(qpsub.interval_flag);
    }
  else
    {
```

```
      exit_status = 1;
      goto END;
    }

 END:
  return exit_status;
}


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

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

  return (x > 0.0)?1.0/sqrt(x):0.0;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_1d_quad_inf_wt_trig_1 (d01ssc) Example Program Results
(User-supplied callback g, first invocation.)
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
```