NAG C Library Function Document

nag_1d_quad_brkpts_1 (d01slc)

1 Purpose

nag_1d_quad_brkpts_1 (d01slc) is a general purpose integrator which calculates an approximation to the integral of a function \( f(x) \) over a finite interval \([a, b]\):

\[
I = \int_{a}^{b} f(x) \, dx.
\]

where the integrand may have local singular behaviour at a finite number of points within the integration interval.

2 Specification

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

void nag_1d_quad_brkpts_1 (double (*f)(double x, Nag_User *comm),
    double a, double b, Integer nbrkpts, double brkpts[], double epsabs,
    double epsrel, Integer max_num_subint, double *result,
    double *abserr, NAG_QquadProgress *qp, Nag_User *comm, NagError *fail)
```

3 Description

This function is based upon the QUADPACK routine QAGP (Piessens et al. (1983)). It is very similar to nag_1d_quad_gen_1 (d01sje), but allows the user to supply ‘break-points’, points at which the function is known to be difficult. It is an adaptive routine, using the Gauss 10-point and Kronrod 21-point rules. The algorithm described by De Doncker (1978), incorporates a global acceptance criterion (as defined by Malcolm and Simpson (1976)) together with the \(c\)-algorithm (Wynn (1956)) to perform extrapolation. The user-supplied ‘break-points’ always occur as the end-points of some sub-interval during the adaptive process. The local error estimation is described by Piessens et al. (1983).

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:

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

1: \( x \) – double

*Input*

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

2: \( \text{comm} \) – Nag_User *

*Input/Output*

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

\( p \) – Pointer

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

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

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

   On entry: the upper limit of integration, b. It is not necessary that a < b.

4:  nbrkpts – Integer

   On entry: the number of user-supplied break-points within the integration interval.

   Constraint: nbrkpts ≥ 0.

5:  brkpts[nbrkpts] – double

   On entry: the user-specified break-points.

   Constraint: the break-points must all lie within the interval of integration (but may be supplied in any order).

6:  epsabs – double

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

7:  epsrel – double

   On entry: the relative accuracy required. If epsrel is negative, the absolute value is used. See Section 6.1.

8:  max_num_subint – Integer

   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.

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

   Constraint: max_num_subint ≥ 1.

9:  result – double *

   On exit: the approximation to the integral I.

10: abserr – double *

    On exit: an estimate of the modulus of the absolute error, which should be an upper bound for |I−result|.

11: qp – Nag_QuadProgress *

    Pointer to structure of type Nag_QuadProgress with the following members:

       num_subint – Integer

       On exit: the actual number of sub-intervals used.

       fun_count – Integer

       On exit: the number of function evaluations performed by nag_1d_quad_brkpts_1.
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_2_INT_ARG_LE or NE_ALLOC_FAIL occurs, these arrays will contain information which may be useful. For details, see Section 6.

Before a subsequent call to nag_l1_quad_bkpts_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 NAG_FREE.

12: comm – Nag_User *

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

p – Pointer Input/Output

On entry/on exit: the pointer p, of type Pointer, allows the user to communicate information to and from the user-defined function f(). 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 is void *.

13: fail – NagError *

Input/Output

The NAG error parameter (see the Essential Introduction).

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

5 Error Indicators and Warnings

NE_INT_ARG_LT

On entry, max_num_subint must not be less than 1: max_num_subint = <value>.
On entry, nbkpts must not be less than 0: nbkpts = <value>.

NE_2_INT_ARG_LE

On entry, max_num_subint = <value> while nbkpts = <value>. These parameters must satisfy max_num_subint > nbkpts.

NE_ALLOC_FAIL

Memory allocation failed.

NE_QUAD_MAX_SUBDIV

The maximum number of subdivisions has been reached: 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 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 = <value>, epsrel = <value>.
The error may be underestimated. Consider relaxing the accuracy requirements specified by epsabs and epsrel.
NE QUAD BAD_SUBDIV

Extremely bad integrand behaviour occurs around the sub-interval \(<value>, <value>\).
The same advice applies as in the case of \texttt{NE QUAD MAX_SUBDIV}.

NE QUAD ROUND OFF 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 \texttt{NE QUAD MAX_SUBDIV}.

NE QUAD NO_CONV

The integral is probably divergent, or slowly convergent.
Please note that divergence can occur with any error exit other than \texttt{NE\_INT\_ARG\_LT},
\texttt{NE\_2\_INT\_ARG\_LE} and \texttt{NE\_ALLOC\_FAIL}.

NE QUAD BRKPTSINVAL

On entry, break points outside \((a, b)\): \(a = <value>, b = <value>\).

6 Further Comments

The function taken by \texttt{nag\_1d\_quad\_brkpts\_1} depends on the integrand and the accuracy required.

If the function fails with an error exit other than \texttt{NE\_INT\_ARG\_LT}, \texttt{NE\_2\_INT\_ARG\_LE} or
\texttt{NE\_ALLOC\_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\_brkpts\_1} along with the integral contributions and
error estimates over the 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, \( \int_a^b f(x)dx \approx r_i \) and \texttt{result} = \( \sum_{i=1}^{n} r_i \) unless the function terminates while testing for divergence of
the integral (see Section 3.4.3 of Piessens et al. (1983)). In this case, \texttt{result} (and \texttt{abserr}) are taken to be
the values returned from the extrapolation process. The value of \(n\) is returned in \texttt{num\_subint}, and the
values \(a_i, b_i, r_i\) and \(e_i\) are stored in the structure \texttt{qp} as

\[
\begin{align*}
  a_i &= \texttt{sub\_int\_beg\_pts}[i-1], \\
  b_i &= \texttt{sub\_int\_end\_pts}[i-1], \\
  r_i &= \texttt{sub\_int\_result}[i-1] \text{ and} \\
  e_i &= \texttt{sub\_int\_error}[i-1].
\end{align*}
\]

6.1 Accuracy

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

\[ |I - \texttt{result}| \leq \texttt{tol} \]

where

\[ \texttt{tol} = \max\{|\texttt{epsabs}|, |\texttt{epsrel}| \times |I|\} \]

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

\[ |I - \texttt{result}| \leq \texttt{abserr} \leq \texttt{tol}. \]

6.2 References

13 (2) 12–18


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_gen_1 (d01sjc)
nag_1d_quad_osc_1 (d01skc)

8 Example

To compute

$$\int_0^1 \frac{1}{\sqrt{|x - \frac{1}{4}|}} \, dx.$$  

8.1 Program Text

/* nag_1d_quad_brkpts_1(d01slc) Example Program */
* Mark 6 revised, 2000.
*/

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

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

main()
{
  double a, b;
  double epsabs, abser, epsrel, brkpts[1], result;
  Integer nbrkpts;
  Nag_QuadProgress qp;
  Integer max_num_subint;
  static NagError fail;
  Nag_User comm;

  Vprintf("d01slc Example Program Results\n");
  nbrkpts = 1;
  epsabs = 0.0;
  epsrel = 0.001;
  a = 0.0;
  b = 1.0;
  max_num_subint = 200;
  brkpts[0] = 1.0/7.0;
d01slc(f, a, b, nbrkpts, brkpts, epsabs, epsrel, max_num_subint,
   &result, &abserr, &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", epsrel);
Vprintf("brkpts[0] - given break-point = %10.4f\n", brkpts[0]);
if (fail.code != NE_NOERROR)
  Vprintf("\%s\n", fail.message);
if (fail.code != NE_INT_ARG_LT && fail.code != NE_2_INT_ARG_LE &&
   fail.code != NE_ALLOC_FAIL)
  {
    /* 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);
  }
if (fail.code != NE_INT_ARG_LT && fail.code != NE_2_INT_ARG_LE
   && fail.code != NE_QUAD_BRKPTS_INVAL && fail.code != NE_ALLOC_FAIL)
  {
    Vprintf("result - approximation to the integral = %9.5f\n", result);
    Vprintf("abseerr - estimate of the absolute error = %9.2e\n", abseerr);
    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);
    exit(EXIT_SUCCESS);
  }
exit(EXIT_FAILURE);
}

static double f(double x, Nag_User *comm)
{
  double a;
  a = FABS(x-1.0/7.0);
  return (a != 0.0) ? pow(a, -0.5) : 0.0;
}

8.2 Program Data

None.

8.3 Program Results

D01SLC Example Program Results
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-03
brkpts[0] - given break-point = 0.1429
result - approximation to the integral = 2.60757
abseerr - estimate of the absolute error = 5.46e-14
qp.fun_count - number of function evaluations = 462
qp.num_subint - number of subintervals used = 12