NAG 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

```
#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, const double brkpts[],
     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_brkpts_1 (d01slc) is based upon the QUADPACK routine QAGP (Piessens *et al.* (1983)). It is very similar to nag_1d_quad_gen_1 (d01sjc), but allows you to supply 'break-points', points at which the function is known to be difficult. It is an adaptive function, 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 ϵ-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 References

de Doncker E (1978) An adaptive extrapolation algorithm for automatic integration *ACM SIGNUM Newsl.* **13(2)** 12–18

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: **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: **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: **b** – double *Input*

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

4: **nbrkpts** – Integer *Input*

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

Constraint: **nbrkpts** ≥ 0.

5: **brkpts**[**nbrkpts**] – const double *Input*

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 *Input*

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

7: **epsrel** – double *Input*

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

8: **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.

9: **result** – double * *Output*

On exit: the approximation to the integral I.

10: **abserr** – double * *Output*

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

11: **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_brkpts_1 (d01slc).

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 9.

Before a subsequent call to nag_1d_quad_brkpts_1 (d01slc) 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`.

12: **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 *`.

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

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

6 Error Indicators and Warnings

NE_2_INT_ARG_LE

On entry, **max_num_subint** $= \langle value \rangle$ while **nbrkpts** $= \langle value \rangle$. These arguments must satisfy **max_num_subint** $>$ **nbrkpts**.

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** $= \langle value \rangle$.

On entry, **nbrkpts** $= \langle value \rangle$.
Constraint: **nbrkpts** ≥ 0.

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_BRKPTS_INVAL

On entry, break-points outside (**a**, **b**): **a** $= \langle value \rangle$, **b** $= \langle value \rangle$.

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_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_2_INT_ARG_LE and NE_ALLOC_FAIL.

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

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_brkpts_1 (d01slc) cannot guarantee, but in practice usually achieves, the following accuracy:

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

where

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

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

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

8 Parallelism and Performance

Not applicable.

9 Further Comments

The time taken by nag_1d_quad_brkpts_1 (d01slc) depends on the integrand and the accuracy required.

If the function fails with an error exit other than NE_INT_ARG_LT, NE_2_INT_ARG_LE 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_brkpts_1 (d01slc) along with the integral contributions and error estimates over the sub-intervals.

Specifically, $i = 1, 2, \ldots n$, 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** $= \sum_{i=1}^{n} r_i$ unless the function terminates while testing for divergence of the integral (see Section 3.4.3 of Piessens *et al.* (1983)). In this case, **result** (and **abserr**) are taken to be the values returned from the extrapolation process. 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

$a_i =$ **qp→sub_int_beg_pts**$[i-1]$,

$b_i =$ **qp→sub_int_end_pts**$[i-1]$,

$r_i =$ **qp→sub_int_result**$[i-1]$ and

$e_i =$ **qp→sub_int_error**$[i-1]$.

10 Example

This example computes

$$\int_0^1 \frac{1}{\sqrt{\left|x - \frac{1}{7}\right|}} dx.$$

10.1 Program Text

```
/* nag_1d_quad_brkpts_1 (d01slc) 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>

#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, brkpts[1], result;
  Integer          nbrkpts;
  Nag_QuadProgress qp;
  Integer          max_num_subint;
  NagError         fail;
  Nag_User         comm;

  INIT_FAIL(fail);

  printf("nag_1d_quad_brkpts_1 (d01slc) Example Program Results\n");
```

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

  nbrkpts = 1;
  epsabs = 0.0;
  epsrel = 0.001;
  a = 0.0;
  b = 1.0;
  max_num_subint = 200;
  brkpts[0] = 1.0/7.0;
  /* nag_1d_quad_brkpts_1 (d01slc).
   * One-dimensional adaptive quadrature, allowing for
   * singularities at specified points, thread-safe
   */
  nag_1d_quad_brkpts_1(f, a, b, nbrkpts, brkpts, 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);
  printf("brkpts[0] - given break-point = %10.4f\n", brkpts[0]);
  if (fail.code != NE_NOERROR)
    printf("Error from nag_1d_quad_brkpts_1 (d01slc) %s\n",
           fail.message);
  if (fail.code != NE_INT_ARG_LT && fail.code != NE_2_INT_ARG_LE &&
      fail.code != NE_ALLOC_FAIL && fail.code != NE_NO_LICENCE)
    {
      /* 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
     && 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);
    }
  else
    {
      exit_status = 1;
      goto END;
    }

 END:
  return exit_status;
}

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

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

```
    }

  a = FABS(x-1.0/7.0);
  return (a != 0.0)?pow(a, -0.5):0.0;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_1d_quad_brkpts_1 (d01slc) Example Program Results
(User-supplied callback f, first invocation.)
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
abserr - estimate of the absolute error =    5.51e-14
qp.fun_count  - number of function evaluations =  462
qp.num_subint  - number of subintervals used =   12
```