NAG Library Function Document

nag_multid_quad_adapt_1 (d01wcc)

1 Purpose

nag_multid_quad_adapt_1 (d01wcc) attempts to evaluate a multidimensional integral (up to 15 dimensions), with constant and finite limits,

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2}\cdots\int_{a_n}^{b_n} f(x_1,x_2,\ldots,x_n)dx_n\cdots dx_2dx_1$$

to a specified relative accuracy, using an adaptive subdivision strategy.

2 Specification

```
#include <nag.h>
#include <nagd01.h>
void nag_multid_quad_adapt_1 (Integer ndim,
     double (*f)(Integer ndim, const double x[], Nag_User *comm),
     const double a[], const double b[], Integer *minpts, Integer maxpts,
     double eps, double *finval, double *acc, Nag_User *comm, NagError *fail)
```

3 Description

nag_multid_quad_adapt_1 (d01wcc) evaluates an estimate of a multidimensional integral over a hyper-rectangle (i.e., with constant limits), and also an estimate of the relative error. You will need to set the relative accuracy required, supply the integrand as a function **f**, and also set the minimum and maximum acceptable number of calls to **f** (in **minpts** and **maxpts**).

The function operates by repeated subdivision of the hyper-rectangular region into smaller hyper-rectangles. In each subregion, the integral is estimated using a seventh-degree rule, and an error estimate is obtained by comparison with a fifth-degree rule which uses a subset of the same points. The fourth differences of the integrand along each coordinate axis are evaluated, and the subregion is marked for possible future subdivision in half along that coordinate axis which has the largest absolute fourth difference.

If the estimated errors, totalled over the subregions, exceed the requested relative error (or if fewer than **minpts** calls to **f** have been made), further subdivision is necessary, and is performed on the subregion with the largest estimated error, that subregion being halved along the appropriate coordinate axis.

The function will fail if the requested relative error level has not been attained by the time **maxpts** calls to **f** have been made.

This function is based on the HALF subroutine developed by van Dooren and de Ridder (1976). It uses a different basic rule, described by Genz and Malik (1980).

4 References

Genz A C and Malik A A (1980) An adaptive algorithm for numerical integration over an N-dimensional rectangular region *J. Comput. Appl. Math.* **6** 295–302

van Dooren P and de Ridder L (1976) An adaptive algorithm for numerical integration over an N-dimensional cube *J. Comput. Appl. Math.* **2** 207–217

5 Arguments

1: **ndim** – Integer *Input*

On entry: the number of dimensions of the integral, n.

Constraint: $2 \leq \mathbf{ndim} \leq 15$.

2: **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 (Integer ndim, const double x[], Nag_User *comm)
```

1: **ndim** – Integer *Input*

On entry: the number of dimensions of the integral.

2: **x**[**ndim**] – const double *Input*

On entry: the coordinates of the point at which the integrand must be evaluated.

3: **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.)

3: **a**[**ndim**] – const double *Input*

On entry: the lower limits of integration, a_i, for $i = 1, 2, \ldots, n$.

4: **b**[**ndim**] – const double *Input*

On entry: the upper limits of integration, b_i, for $i = 1, 2, \ldots, n$.

5: **minpts** – Integer * *Input/Output*

On entry: **minpts** must be set to the minimum number of integrand evaluations to be allowed.

On exit: **minpts** contains the actual number of integrand evaluations used by this function.

6: **maxpts** – Integer *Input*

On entry: the maximum number of integrand evaluations to be allowed.

Constraints:

$\mathbf{maxpts} \geq \mathbf{minpts}$;
$\mathbf{maxpts} \geq 2^{\mathbf{ndim}} + 2 \times \mathbf{ndim}^2 + 2 \times \mathbf{ndim} + 1$.

7: **eps** – double *Input*

On entry: the relative error acceptable. When the solution is zero or very small relative accuracy may not be achievable but you may still set **eps** to a reasonable value and check **fail** for NE_QUAD_MAX_INTEGRAND_EVAL.

Constraint: $\mathbf{eps} > 0.0$.

8: **finval** – double * *Output*

On exit: the best estimate obtained for the integral.

9: **acc** – double * *Output*

On exit: the estimated relative error in **finval**.

10: **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 will be `void *` with a C compiler that defines `void *` and `char *` otherwise.

11: **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_LT

On entry, **maxpts** $= \langle value \rangle$ while **minpts** $= \langle value \rangle$. These arguments must satisfy **maxpts** $\geq$ **minpts**.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_INVALID_INT_RANGE_2

Value $\langle value \rangle$ given to **ndim** not valid. Correct range is $2 \leq \mathbf{ndim} \leq 15$.

NE_QUAD_MAX_INTEGRAND_CONS

maxpts $< \langle value \rangle$. Constraint: $\mathbf{maxpts} \geq 2^{\mathbf{ndim}} + 2 \times \mathbf{ndim}^2 + 2 \times \mathbf{ndim} + 1$.

NE_QUAD_MAX_INTEGRAND_EVAL

maxpts was too small to obtain the required accuracy. On return, **finval** and **acc** contain estimates of the integral and the relative error, but **acc** will be greater than **eps**.

NE_REAL_ARG_LE

On entry, **eps** must not be less than or equal to 0.0: **eps** $= \langle value \rangle$.

7 Accuracy

A relative error estimate is output through the argument **acc**.

8 Parallelism and Performance

Not applicable.

9 Further Comments

Execution time will usually be dominated by the time taken to evaluate the integrand **f**, and hence the maximum time that could be taken will be proportional to **maxpts**.

10 Example

This example estimates the integral

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{4z_1 z_3^2 \exp(2z_1 z_3)}{(1 + z_2 + z_4)^2} dz_4 dz_3 dz_2 dz_1 = 0.575364.$$

The accuracy requested is one part in 10,000.

10.1 Program Text

```
/* nag_multid_quad_adapt_1 (d01wcc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 * 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(Integer n, const double z[], Nag_User *comm);
#ifdef __cplusplus
}
#endif

#define NDIM   4
#define MAXPTS 1000*NDIM

int main(void)
{
  static Integer use_comm[1] = {1};
  Integer  exit_status = 0;
  Integer  ndim = NDIM;
  Integer  maxpts = MAXPTS;
  double   a[4], b[4];
  Integer  k;
  double   finval;
  Integer  minpts;
  double   acc, eps;
  Nag_User comm;
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_multid_quad_adapt_1 (d01wcc) Example Program Results\n");

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

  for (k = 0; k < 4; ++k)
    {
      a[k] = 0.0;
      b[k] = 1.0;
    }
```

```
  eps = 0.0001;
  minpts = 0;

  /* nag_multid_quad_adapt_1 (d01wcc).
   * Multi-dimensional adaptive quadrature, thread-safe
   */
  nag_multid_quad_adapt_1(ndim, f, a, b, &minpts, maxpts, eps, &finval, &acc,
                          &comm, &fail);

  if (fail.code != NE_NOERROR && fail.code != NE_QUAD_MAX_INTEGRAND_EVAL)
    {
      printf("Error from nag_multid_quad_adapt_1 (d01wcc) %s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  printf("Requested accuracy =%12.2e\n", eps);
  printf("Estimated value    =%12.4f\n", finval);
  printf("Estimated accuracy =%12.2e\n", acc);

 END:
  return exit_status;
}


static double NAG_CALL f(Integer n, const double z[], Nag_User *comm)
{
  double tmp_pwr;
  Integer *use_comm = (Integer *)comm->p;

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

  tmp_pwr = z[1]+1.0+z[n-1];
  return z[0]*4.0*z[2]*z[2]*exp(z[0]*2.0*z[2])/(tmp_pwr*tmp_pwr);
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_multid_quad_adapt_1 (d01wcc) Example Program Results
(User-supplied callback f, first invocation.)
Requested accuracy =    1.00e-04
Estimated value    =      0.5754
Estimated accuracy =    9.89e-05
```