NAG C Library Function Document

nag_multid_quad_monte_carlo (d01gbc)

1 Purpose

nag_multid_quad_monte_carlo (d01gbc) evaluates an approximation to the integral of a function over a hyper-rectangular region, using a Monte Carlo method. An approximate relative error estimate is also returned. This routine is suitable for low accuracy work.

2 Specification

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

void nag_multid_quad_monte_carlo (Integer ndim,
    double (*f)(Integer ndim, double x[]),
    Nag_MCMethd method, Nag_Start cont, double a[], double b[],
    Integer *mincsl, Integer maxcsl, double eps, double *finit,
    double *acc, double **comm_arr, NagError *fail)
```

3 Description

This function uses an adaptive Monte Carlo method based on the algorithm described by Lautrup (1971). It is implemented for integrals of the form:

$$
\int_{a_1}^{b_1} \int_{a_2}^{b_2} \ldots \int_{a_n}^{b_n} f(x_1, x_2, \ldots, x_n) \, dx_1 \ldots dx_n
$$

Upon entry, unless the parameter method has the value Nag_OneIteration, the routine subdivides the integration region into a number of equal volume subregions. Inside each subregion the integral and the variance are estimated by means of pseudo-random sampling. All contributions are added together to produce an estimate for the whole integral and total variance. The variance along each co-ordinate axis is determined and the routine uses this information to increase the density and change the widths of the sub-intervals along each axis, so as to reduce the total variance. The total number of subregions is then increased by a factor of two and the program recycles for another iteration. The program stops when a desired accuracy has been reached or too many integral evaluations are needed for the next cycle.

4 Parameters

1. **ndim** – Integer
   
   *Input*

   On entry: the number of dimensions of the integral, \( n \).

   Constraint: \( ndim \geq 1 \).

2. **f** – function supplied by user
   
   *Function*

   The function \( f \), supplied by the user, must return the value of the integrand \( f \) at a given point.

   The specification of \( f \) is:
double f(Integer ndim, double x[])

1:  

   ndim – Integer
   
   On entry: the number of dimensions of the integral.

2:  

   x[ndim] – double
   
   On entry: the co-ordinates of the point at which the integrand must be evaluated.

3:  

   method – Nag_MCMETHOD
   
   On entry: the method to be used.
   
   If method = Nag_OneIteration, then the function uses only one iteration of a crude Monte Carlo method with maxcls sample points.
   
   If method = Nag_ManyIterations, then the function subdivides the integration region into a number of equal volume subregions.

   Constraint: method = Nag_OneIteration or Nag_ManyIterations.

4:  

   cont – Nag_Start
   
   On entry: the continuation state of the evaluation of the integrand.
   
   If cont = Nag_Cold, indicates that this is the first call to the function with the current integrand and parameters ndim, a and b.
   
   If cont = Nag_Hot, indicates that a previous call has been made with the same parameters ndim, a and b with the same integrand. Please note that method must not be changed.
   
   If cont = Nag_Warm, indicates that a previous call has been made with the same parameters ndim, a and b but that the integrand is new. Please note that method must not be changed.

   Constraint: cont = Nag_Cold, Nag_Warm or Nag_Hot.

5:  

   a[ndim] – double
   
   On entry: the lower limits of integration, aᵢ, for i = 1, 2, ..., n.

6:  

   b[ndim] – double
   
   On entry: the upper limits of integration, bᵢ, for i = 1, 2, ..., n.

7:  

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

   Constraint: 0 ≤ mincls < maxcls.

   On exit: mincls contains the total number of integrand evaluations actually used by nag_multid_quad_monte_carlo.

8:  

   maxcls – Integer
   
   On entry: the maximum number of integrand evaluations to be allowed. In the continuation case this is the number of new integrand evaluations to be allowed. These counts do not include zero integrand values.

   Constraints:
   
   maxcls ≥ 4 × (ndim + 1).
9: **eps** – double

*Input*

*On entry:* the relative accuracy required.

*Constraint:* $\text{eps} \geq 0.0$.

10: **finest** – double *

*Output*

*On exit:* the best estimate obtained for the integral.

11: **acc** – double *

*Output*

*On exit:* the estimated relative accuracy of *finest*.

12: **comm_arr** – double **

*Input/Output*

*On entry:* if *cont* = **Nag_Warm** or **Nag_Hot**, the memory pointed to and allocated by a previous call of **nag_multid_quad_monte_carlo** must be unchanged.

If *cont* = **Nag_Cold** then appropriate memory is allocated internally by **nag_multid_quad_monte_carlo**.

*On exit:* **comm_arr** contains information about the current sub-interval structure which could be used in later calls of **nag_multid_quad_monte_carlo**. In particular, **comm_arr**[$j-1$] gives the number of sub-intervals used along the $j$th co-ordinate axis.

When this information is no longer useful, or before a subsequent call to **nag_multid_quad_monte_carlo** with *cont* = **Nag_Cold** is made, the user should free the storage contained in this pointer using the NAG macro **NAG_FREE**. Note this memory will have been allocated and needs to be freed only if the error exit **NE_NOERROR** or **NE_QUAD_MAX_INTEGRAND_EVAL** occurs. Otherwise, no memory needs to be freed.

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_LE**

On entry, **mincl** must not be less than or equal to 0: **mincl** = <value>.

**NE_INT_ARG_LT**

On entry, **ndim** must not be less than 1: **ndim** = <value>.

**NE_REAL_ARG_LT**

On entry, **eps** must not be less than 0.0: **eps** = <value>.

**NE_2_INT_ARG_GE**

On entry, **mincl** = <value> while **maxcl** = <value>. These parameters must satisfy **mincl** < **maxcl**.

**NE_2_INT_ARG_LT**

On entry, **maxcl** = <value> while **ndim** = <value>. These parameters must satisfy **maxcl** $\geq 4 \times (\text{ndim}+1)$.

**NE_BAD_PARAM**

On entry, parameter **method** had an illegal value.

On entry, parameter **cont** had an illegal value.
NE_QUAD_MAX_INTEGRAND_EVAL

maxcls was too small to obtain the required accuracy.
In this case nag_multid_quad_monte_carlo returns a value of finest with estimated relative error acc,
but acc will be greater than eps. This error exit may be taken before maxcls non-zero integrand
evaluations have actually occurred, if the routine calculates that the current estimates could not be
improved before maxcls was exceeded.

NE_ALLOC_FAIL

Memory allocation failed.

6 Further Comments

The running time for nag_multid_quad_monte_carlo will usually be dominated by the time used to
evaluate the integrand f, so the maximum time that could be used is approximately proportional to maxcls.

For some integrands, particularly those that are poorly behaved in a small part of the integration region,
this function may terminate with a value of acc which is significantly smaller than the actual relative error.
This should be suspected if the returned value of mincls is small relative to the expected difficulty of the
integral. Where this occurs, nag_multid_quad_monte_carlo should be called again, but with a higher entry
value of mincls (e.g., twice the returned value) and the results compared with those from the previous call.
The exact values of finest and acc on return will depend (within statistical limits) on the sequence of
random numbers generated within this function by calls to nag_random_continuous_uniform (g05cac).
Separate runs will produce identical answers unless the part of the program executed prior to calling this
function also calls (directly or indirectly) routines from Chapter g05, and the series of such calls differs
between runs. If desired, the user may ensure the identity or difference between runs of the results returned
by this function, by calling nag_random_init_repeatable (g05cbc) or nag_random_init_nonrepeatable
(g05ccc) respectively, immediately before calling this function.

6.1 Accuracy

A relative error estimate is output through the parameter acc. The confidence factor is set so that the actual
error should be less than acc 90% of the time. If a user desires a higher confidence level then a smaller
value of eps should be used.

6.2 References

in Theoretical Physics, Marseille

7 See Also

nag_multid_quad_adapt (d01fcc)

8 Example

This example program calculates the integral

\[ \int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{4x_1x_2^2\exp(2x_1x_3)}{(1 + x_2 + x_4)^2} dx_1 dx_2 dx_3 dx_4 = 0.575364. \]
8.1 Program Text

/* nag_multid_quad_monte_carlo(d01gbc) Example Program */
/* Copyright 1991 Numerical Algorithms Group. */
/* Mark 2, 1991. */
/* Mark 6 revised, 2000. */

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

static double f(Integer ndim, double x[]);

#define NDIM 4
#define MAXCLSL 20000

main()
{
  double a[4], b[4];
  Integer k, mincls;
  double finest;
  double acc, eps;
  Integer ndim = NDIM;
  Integer maxcls = MAXCLSL;
  static NagError fail;
  double *comm_arr = (double *)0;
  Nag_MCMетод method;

  Vprintf("d01gbc Example Program Results\n");
  for (k=0; k<4; ++k)
  {
    a[k] = 0.0;
    b[k] = 1.0;
  }
  eps = 0.01;
  mincls = 1000;
  method = Nag_ManyIterations;
  cont = Nag_Cold;
  d01gbc(ndim, f, method, cont, a, b, &mincls, maxcls, eps,
       &finest, &acc, &comm_arr, &fail);
  if (fail.code != NE_NOERROR)
    Vprintf("%s\n", fail.message);
  if (fail.code == NE_NOERROR || fail.code == NE_QUAD_MAX_INTEGRAND_EVAL)
  {
    Vprintf("Requested accuracy = %10.2e\n", eps);
    Vprintf("Estimated value = %10.5f\n", finest);
    Vprintf("Estimated accuracy = %10.2e\n", acc);
    Vprintf("Number of evaluations = %5d\n", mincls);
    /* Free memory allocated internally */
    NAG_FREE(comm_arr);
    exit(EXIT_SUCCESS);
  }
}
```c

derived
    exit(EXIT_FAILURE);
}

static double f(Integer ndim, double x[])
{
    return x[0]*4.0*(x[2]*x[2])*exp(x[0]*2.0*x[2])/((x[1]+1.0+x[3])*(x[1]+1.0+x[3]));
}

8.2 Program Data
None.

8.3 Program Results

<table>
<thead>
<tr>
<th>Example Program Results</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Requested accuracy</td>
<td>1.00e-02</td>
</tr>
<tr>
<td>Estimated value</td>
<td>0.57554</td>
</tr>
<tr>
<td>Estimated accuracy</td>
<td>8.20e-03</td>
</tr>
<tr>
<td>Number of evaluations</td>
<td>1728</td>
</tr>
</tbody>
</table>
```