NAG C Library Function Document
nag_multid_quad_monte_carlo_1 (d01xbc)

1 Purpose
nag_multid_quad_monte_carlo_1 (d01xbc) 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

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

void nag_multid_quad_monte_carlo_1 (Integer ndim,
 double (*f)(Integer ndim, double x[], Nag_User *comm),
 Nag_MCMeth method, Nag_Stat cont, double a[], double b[],
 Integer *mncls, Integer maxcls, double eps, double *finest,
 double *acc, double **comm_arr, NAG_User *comm, 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: \textbf{ndim} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: the number of dimensions of the integral, \textit{n}.

\textit{Constraint}: \textit{ndim} \geq 1.

2: \textbf{f} – function supplied by user \hspace{1cm} \textit{Function}

The function \textit{f}, supplied by the user, must return the value of the integrand \textit{f} at a given point.

The specification of \textit{f} is:

\[
\text{double f(Integer ndim, double x[], Nag_User *comm)}
\]

1: \textbf{ndim} – Integer \hspace{1cm} \textit{Input}

\textit{On entry}: the number of dimensions of the integral.
2: \( \text{\texttt{x[ndim]}} \) – double

\textit{Input}

\textit{On entry:} the co-ordinates of the point at which the integrand must be evaluated.

3: \( \text{\texttt{comm}} \) – \texttt{Nag_User} *

\textit{On entry/on exit:} pointer to a structure of type \texttt{Nag_User} with the following member:

\( \text{\texttt{p}} \) – Pointer

\textit{Input/Output}

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

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

to obtain the original object’s address with appropriate type. (See the argument \( \text{\texttt{comm}} \) below.)

3: \( \text{\texttt{method}} \) – \texttt{Nag_MCMeth}od

\textit{Input}

\textit{On entry:} the method to be used.

\textit{If \texttt{method} = \texttt{Nag_OnelIteration}, then the function uses only one iteration of a crude Monte Carlo method with \texttt{maxcls} sample points.}

\textit{If \texttt{method} = \texttt{Nag_ManyIterations}, then the function subdivides the integration region into a number of equal volume subregions.}

\textit{Constraint: \texttt{method} = \texttt{Nag_OnelIteration} or \texttt{Nag_ManyIterations}.}

4: \( \text{\texttt{cont}} \) – \texttt{Nag_Start}

\textit{Input}

\textit{On entry:} the continuation state of the evaluation of the integrand.

\textit{If \texttt{cont} = \texttt{Nag_Cold}, indicates that this is the first call to the function with the current integrand and parameters \texttt{ndim}, \texttt{a} and \texttt{b}.}

\textit{If \texttt{cont} = \texttt{Nag_Hot}, indicates that a previous call has been made with the same parameters \texttt{ndim}, \texttt{a} and \texttt{b} with the same integrand. Please note that \texttt{method} must not be changed.}

\textit{If \texttt{cont} = \texttt{Nag_Warm}, indicates that a previous call has been made with the same parameters \texttt{ndim}, \texttt{a} and \texttt{b} but that the integrand is new. Please note that \texttt{method} must not be changed.}

\textit{Constraint: \texttt{cont} = \texttt{Nag_Cold}, \texttt{Nag_Warm} or \texttt{Nag_Hot}.}

5: \( \text{\texttt{a[ndim]}} \) – double

\textit{Input}

\textit{On entry:} the lower limits of integration, \( a_i \), for \( i = 1, 2, \ldots, n \).

6: \( \text{\texttt{b[ndim]}} \) – double

\textit{Input}

\textit{On entry:} the upper limits of integration, \( b_i \), for \( i = 1, 2, \ldots, n \).

7: \( \text{\texttt{mincls}} \) – Integer *

\textit{Input/Output}

\textit{On entry:} \( \text{\texttt{mincls}} \) must be set to the minimum number of integrand evaluations to be allowed.

\textit{Constraint:} \( 0 \leq \text{\texttt{mincls}} < \text{\texttt{maxcls}}. \)

\textit{On exit:} \( \text{\texttt{mincls}} \) contains the total number of integrand evaluations actually used by \texttt{nag_multid_quad_monte_carlo_1}. 
8: \texttt{maxcls} – Integer \hspace{1cm} \textit{Input}

\textit{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.

\textit{Constraints:}

\begin{align*}
\texttt{maxcls} &> \texttt{mincls}; \\
\texttt{maxcls} &\geq 4 \times (\texttt{ndim} + 1).
\end{align*}

9: \texttt{eps} – double \hspace{1cm} \textit{Input}

\textit{On entry:} the relative accuracy required.

\textit{Constraint:} \texttt{eps} \geq 0.0.

10: \texttt{finest} – double * \hspace{1cm} \textit{Output}

\textit{On exit:} the best estimate obtained for the integral.

11: \texttt{acc} – double * \hspace{1cm} \textit{Output}

\textit{On exit:} the estimated relative accuracy of \texttt{finest}.

12: \texttt{comm\_arr} – double ** \hspace{1cm} \textit{Input/Output}

\textit{On entry:} if \texttt{cont} = \texttt{Nag\_Warm} or \texttt{Nag\_Hot}, the memory pointed to and allocated by a previous call of \texttt{lag\_multid\_quad\_monte\_carlo\_1} must be unchanged.

If \texttt{cont} = \texttt{Nag\_Cold} then appropriate memory is allocated internally by \texttt{lag\_multid\_quad\_monte\_\_carlo\_1}.

\textit{On exit:} \texttt{comm\_arr} contains information about the current sub-interval structure which could be used in later calls of \texttt{lag\_multid\_quad\_monte\_carlo\_1}. In particular, \texttt{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 \texttt{lag\_multid\_quad\_monte\_\_carlo\_1} with \texttt{cont} = \texttt{Nag\_Cold} is made, the user should free the storage contained in this pointer using the NAG macro \texttt{NAG\_FREE}. Note this memory will have been allocated and needs to be freed only if the error exit \texttt{NE\_NOERROR} or \texttt{NE\_QUAD\_MAX\_INTEGRAND\_EVAL} occurs. Otherwise, no memory needs to be freed.

13: \texttt{comm} – \texttt{Nag\_User} *

\textit{On entry/on exit:} pointer to a structure of type \texttt{Nag\_User} with the following member:

\begin{itemize}
\item \texttt{p} – Pointer \hspace{1cm} \textit{Input/Output}
\end{itemize}

\textit{On entry/on exit:} the pointer \texttt{p}, of type Pointer, allows the user to communicate information to and from the user-defined function \texttt{f}. An object of the required type should be declared by the user, e.g., a structure, and its address assigned to the pointer \texttt{p} by means of a cast to Pointer in the calling program, e.g., \texttt{comm.p} = (Pointer)&s. The type Pointer is void *.

14: \texttt{fail} – \texttt{NagError} * \hspace{1cm} \textit{Input/Output}

The NAG error parameter (see the Essential Introduction).

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

5 \hspace{1cm} \textbf{Error Indicators and Warnings}

\texttt{NE\_INT\_ARG\_LE}

\textit{On entry, mincls} must not be less than or equal to 0: \texttt{mincls} = \texttt{<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, mincls = <value> while maxcls = <value>.
   These parameters must satisfy mincls < maxcls.

NE_2_INT_ARG_LT
   On entry, maxcls = <value> while ndim = <value>.
   These parameters must satisfy maxcls ≥ 4×(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_1 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_1 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_1 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_repeateable (g05cbc) or nag_random_init_nonrepeateable (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

7 See Also
nag_multid_quad_adapt_1 (d01wcc)

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_1(d01xbc) Example Program */
/* Copyright 1998 Numerical Algorithms Group. */
/* Mark 5, 1998. */
/* Mark 6 revised, 2000. */

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

static double f(Integer ndim, double x[], Nag_User *comm);
#define NDIM 4
#define MAXCLS 20000

main()
{
    double a[4], b[4];
    Integer k, mincls;
    double finest;
    double acc, eps;
    Integer ndim = NDIM;
    Integer maxcls = MAXCLS;
    static NagError fail;
    double *comm_arr = (double *)0;
    Nag_MCMeta method;
    Nag_Start cont;
    Nag_User comm;

    Vprintf("d01xbc 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;

d01xbc(ndim, f, method, cont, a, b, &mincls, maxcls, eps, 
  &finest, &acc, &comm_arr, &comm, &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 = %5ld\n", mincls);
    /* Free memory allocated internally */
    NAG_FREE(comm_arr);
    exit(EXIT_SUCCESS);
  } 
else 
  exit(EXIT_FAILURE);

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

8.2 Program Data
None.

8.3 Program Results

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>d01xbc Example Program Results</td>
<td></td>
</tr>
<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>

---

```
```