NAG Library Function Document

nag_quad_md_numth_vec (d01gdc)

1 Purpose

nag_quad_md_numth_vec (d01gdc) calculates an approximation to a definite integral in up to 20 dimensions, using the Korobov–Conroy number theoretic method.

2 Specification

```
#include <nag.h>
#include <nagd01.h>
void nag_quad_md_numth_vec (Integer ndim,
     void  (*vecfun)(Integer ndim, const double x[], double fv[], Integer m,
           Nag_Comm *comm),
     void  (*vecreg)(Integer ndim, const double x[], Integer j, double c[],
           double d[], Integer m, Nag_Comm *comm),
     Integer npts, double vk[], Integer nrand, Nag_Boolean transform,
     double *res, double *err, Nag_Comm *comm, NagError *fail)
```

3 Description

nag_quad_md_numth_vec (d01gdc) calculates an approximation to the integral

$$I = \int_{c_1}^{d_1} \cdots \int_{c_n}^{d_n} f(x_1, \ldots, x_n)\, dx_n \ldots dx_1 \tag{1}$$

using the Korobov–Conroy number theoretic method (see Korobov (1957), Korobov (1963) and Conroy (1967)). The region of integration defined in (1) is such that generally c_i and d_i may be functions of $x_1, x_2, \ldots, x_{i-1}$, for $i = 2, 3, \ldots, n$, with c_1 and d_1 constants. The integral is first of all transformed to an integral over the n-cube $[0,1]^n$ by the change of variables

$$x_i = c_i + (d_i - c_i) y_i, \quad i = 1, 2, \ldots, n.$$

The method then uses as its basis the number theoretic formula for the n-cube, $[0,1]^n$:

$$\int_0^1 \cdots \int_0^1 g(x_1, \ldots, x_n)\, dx_n \cdots dx_1 = \frac{1}{p} \sum_{k=1}^{p} g\left(\left\{ k \frac{a_1}{p} \right\}, \ldots, \left\{ k \frac{a_n}{p} \right\} \right) - E \tag{2}$$

where $\{x\}$ denotes the fractional part of x, $a_1, \ldots, a_n$ are the so-called optimal coefficients, E is the error, and p is a prime integer. (It is strictly only necessary that p be relatively prime to all $a_1, \ldots, a_n$ and is in fact chosen to be even for some cases in Conroy (1967).) The method makes use of properties of the Fourier expansion of $g(x_1, \ldots, x_n)$ which is assumed to have some degree of periodicity. Depending on the choice of $a_1, \ldots, a_n$ the contributions from certain groups of Fourier coefficients are eliminated from the error, E. Korobov shows that $a_1, \ldots, a_n$ can be chosen so that the error satisfies

$$E \leq CKp^{-\alpha} \ln^{\alpha\beta} p \tag{3}$$

where α and C are real numbers depending on the convergence rate of the Fourier series, β is a constant depending on n, and K is a constant depending on α and n. There are a number of procedures for calculating these optimal coefficients. Korobov imposes the constraint that

$$a_1 = 1 \quad \text{and} \quad a_i = a^{i-1} (\text{mod } p) \tag{4}$$

and gives a procedure for calculating the argument, a, to satisfy the optimal conditions.

In this function the periodisation is achieved by the simple transformation

$$x_i = y_i^2(3 - 2y_i), \quad i = 1, 2, \ldots, n.$$

More sophisticated periodisation procedures are available but in practice the degree of periodisation does not appear to be a critical requirement of the method.

An easily calculable error estimate is not available apart from repetition with an increasing sequence of values of p which can yield erratic results. The difficulties have been studied by Cranley and Patterson (1976) who have proposed a Monte–Carlo error estimate arising from converting (2) into a stochastic integration rule by the inclusion of a random origin shift which leaves the form of the error (3) unchanged; i.e., in the formula (2), $\left\{k\frac{a_i}{p}\right\}$ is replaced by $\left\{\alpha_i + k\frac{a_i}{p}\right\}$, for $i = 1, 2, \ldots, n$, where each α_i, is uniformly distributed over $[0, 1]$. Computing the integral for each of a sequence of random vectors α allows a 'standard error' to be estimated.

This function provides built-in sets of optimal coefficients, corresponding to six different values of p. Alternatively, the optimal coefficients may be supplied by you. Functions nag_quad_md_numth_coeff_prime (d01gyc) and nag_quad_md_numth_coeff_2prime (d01gzc) compute the optimal coefficients for the cases where p is a prime number or p is a product of two primes, respectively.

4 References

Conroy H (1967) Molecular Shroedinger equation VIII. A new method for evaluting multi-dimensional integrals *J. Chem. Phys.* **47** 5307–5318

Cranley R and Patterson T N L (1976) Randomisation of number theoretic methods for mulitple integration *SIAM J. Numer. Anal.* **13** 904–914

Korobov N M (1957) The approximate calculation of multiple integrals using number theoretic methods *Dokl. Acad. Nauk SSSR* **115** 1062–1065

Korobov N M (1963) *Number Theoretic Methods in Approximate Analysis* Fizmatgiz, Moscow

5 Arguments

1: **ndim** – Integer *Input*

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

Constraint: $1 \leq$ **ndim** ≤ 20.

2: **vecfun** – function, supplied by the user *External Function*

vecfun must evaluate the integrand at a specified set of points.

The specification of **vecfun** is:

```
void  vecfun (Integer ndim, const double x[], double fv[], Integer m,
      Nag_Comm *comm)
```

1: **ndim** – Integer *Input*

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

2: **x**[**m** $\times$ **ndim**] – const double *Input*

Note: where $\mathbf{X}(i,j)$ appears in this document, it refers to the array element **x**$[(j-1) \times \mathbf{m} + i - 1]$.

On entry: the coordinates of the m points at which the integrand must be evaluated. $\mathbf{X}(i,j)$ contains the jth coordinate of the ith point.

3: **fv**[**m**] – double *Output*

On exit: **fv**$[i-1]$ must contain the value of the integrand of the ith point, i.e., **fv**$[i-1] = f(\mathbf{X}(i,1), \mathbf{X}(i,2), \ldots, \mathbf{X}(i, \mathbf{ndim}))$, for $i = 1, 2, \ldots, \mathbf{m}$.

4: **m** – Integer *Input*

On entry: the number of points m at which the integrand is to be evaluated.

5: **comm** – Nag_Comm * *Communication Structure*

Pointer to structure of type Nag_Comm; the following members are relevant to **vecfun**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_quad_md_numth_vec (d01gdc) you may allocate memory and initialize these pointers with various quantities for use by **vecfun** when called from nag_quad_md_numth_vec (d01gdc) (see Section 3.2.1.1 in the Essential Introduction).

3: **vecreg** – function, supplied by the user *External Function*

vecreg must evaluate the limits of integration in any dimension for a set of points.

The specification of **vecreg** is:

```
void  vecreg (Integer ndim, const double x[], Integer j, double c[],
      double d[], Integer m, Nag_Comm *comm)
```

1: **ndim** – Integer *Input*

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

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

Note: where $\mathbf{X}(i,j)$ appears in this document, it refers to the array element **x**$[(j-1) \times \mathbf{m} + i - 1]$.

On entry: for $i = 1, 2, \ldots, m$, $\mathbf{X}(i,1)$, $\mathbf{X}(i,2), \ldots, \mathbf{X}(i, j-1)$ contain the current values of the first $(j-1)$ coordinates of the ith point, which may be used if necessary in calculating the m values of c_j and d_j.

3: **j** – Integer *Input*

On entry: the index j for which the limits of the range of integration are required.

4: **c**[**m**] – double *Output*

On exit: **c**$[i-1]$ must be set to the lower limit of the range for $\mathbf{X}(i,j)$, for $i = 1, 2, \ldots, m$.

5: **d**[**m**] – double *Output*

On exit: **d**$[i-1]$ must be set to the upper limit of the range for $\mathbf{X}(i,j)$, for $i = 1, 2, \ldots, m$.

6: **m** – Integer *Input*

On entry: the number of points m at which the limits of integration must be specified.

7: **comm** – Nag_Comm * *Communication Structure*

Pointer to structure of type Nag_Comm; the following members are relevant to **vecreg**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_quad_md_numth_vec (d01gdc) you may allocate memory and initialize these pointers with various quantities for use by **vecreg** when called from nag_quad_md_numth_vec (d01gdc) (see Section 3.2.1.1 in the Essential Introduction).

4: **npts** – Integer *Input*

On entry: the Korobov rule to be used. There are two alternatives depending on the value of **npts**.

(i) $1 \leq$ **npts** ≤ 6.

In this case one of six preset rules is chosen using 2129, 5003, 10007, 20011, 40009 or 80021 points depending on the respective value of **npts** being 1, 2, 3, 4, 5 or 6.

(ii) **npts** > 6.

npts is the number of actual points to be used with corresponding optimal coefficients supplied in the array **vk**.

Constraint: **npts** ≥ 1.

5: **vk**[**ndim**] – double *Input/Output*

On entry: if **npts** > 6, **vk** must contain the n optimal coefficients (which may be calculated using nag_quad_md_numth_coeff_prime (d01gyc) or nag_quad_md_numth_coeff_2prime (d01gzc)).

If **npts** ≤ 6, **vk** need not be set.

On exit: if **npts** > 6, **vk** is unchanged.

If **npts** ≤ 6, **vk** contains the n optimal coefficients used by the preset rule.

6: **nrand** – Integer *Input*

On entry: the number of random samples to be generated (generally a small value, say 3 to 5, is sufficient). The estimate, **res**, of the value of the integral returned by the function is then the average of **nrand** calculations with different random origin shifts. If **npts** > 6, the total number of integrand evaluations will be **nrand** $\times$ **npts**. If $1 \leq$ **npts** ≤ 6, then the number of integrand evaluations will be **nrand** $\times\, p$, where p is the number of points corresponding to the six preset rules. For reasons of efficiency, these values are calculated a number at a time in **vecfun**.

Constraint: **nrand** ≥ 1.

7: **transform** – Nag_Boolean *Input*

On entry: indicates whether the periodising transformation is to be used.

transform = Nag_TRUE
The transformation is to be used.

transform = Nag_FALSE
The transformation is to be suppressed (to cover cases where the integrand may already be periodic or where you want to specify a particular transformation in the definition of **vecfun**).

Suggested value: **transform** = Nag_TRUE.

8: **res** – double * *Output*

On exit: the approximation to the integral I.

9: **err** – double * *Output*

On exit: the standard error as computed from **nrand** sample values. If $\textbf{nrand} = 1$, then **err** contains zero.

10: **comm** – Nag_Comm * *Communication Structure*

The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

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

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

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument $\langle value\rangle$ had an illegal value.

NE_INT

On entry, $\textbf{ndim} = \langle value\rangle$.
Constraint: $1 \leq \textbf{ndim} \leq 20$.

On entry, **npts** must be at least 1: $\textbf{npts} = \langle value\rangle$.

On entry, **nrand** must be at least 1: $\textbf{nrand} = \langle value\rangle$.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

7 Accuracy

If $\textbf{nrand} > 1$, an estimate of the absolute standard error is given by the value, on exit, of **err**.

8 Parallelism and Performance

Not applicable.

9 Further Comments

vecfun and **vecreg** must calculate the integrand and limits of integration at a *set* of points. For some problems the amount of time spent in these two functions, which must be supplied by you, may account for a significant part of the total computation time.

The time taken will be approximately proportional to $\textbf{nrand} \times p$, where p is the number of points used, but may depend significantly on the efficiency of the code provided by you in **vecfun** and **vecreg**.

The exact values of **res** and **err** on return will depend (within statistical limits) on the sequence of random numbers generated within nag_quad_md_numth_vec (d01gdc) by calls to nag_rand_basic (g05sac). Separate runs will produce identical answers.

10 Example

This example calculates the integral

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 \cos(0.5 + 2(x_1 + x_2 + x_3 + x_4) - 4)\, dx_1\, dx_2\, dx_3\, dx_4.$$

10.1 Program Text

```
/* nag_quad_md_numth_vec (d01gdc) Example Program.
 *
 * Copyright 2011, Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd01.h>
#include <nagx04.h>

#ifdef __cplusplus
extern "C" {
#endif
  static void NAG_CALL vecfun(Integer ndim, const double x[], double fv[],
                              Integer m, Nag_Comm *comm);
  static void NAG_CALL vecreg(Integer ndim, const double x[], Integer j,
                              double c[], double d[], Integer m,
                              Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{
  static double ruser[2] = {-1.0, -1.0};
  Integer     exit_status = 0;
  Integer     ndim;
  Integer     npts, nrand;
  double      err, res;
  double     *vk = 0;
  Nag_Boolean transform;
  char        nag_enum_arg[40];
  Nag_Comm comm;
  NagError    fail;

  INIT_FAIL(fail);

  printf("nag_quad_md_numth_vec (d01gdc) Example Program Results\n");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  /* Input parameters */
  scanf("%ld %ld %ld", &ndim, &npts, &nrand);
  /* Nag_Boolean */
  scanf("%39s %*[^\n] ", nag_enum_arg);
  transform = (Nag_Boolean) nag_enum_name_to_value (nag_enum_arg);

  if (!(vk = NAG_ALLOC(ndim, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
```

```
  /* nag_quad_md_numth_vec (d01gdc).
   * Multidimensional quadrature, general product region,
   * number-theoretic method.
   */
  nag_quad_md_numth_vec(ndim, vecfun, vecreg, npts, vk, nrand, transform, &res,
                        &err, &comm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_quad_md_numth_vec (d01gdc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  printf("\nResult = %13.5f, standard error = %10.2e\n", res, err);

 END:
  NAG_FREE(vk);

  return exit_status;
}

static void NAG_CALL vecfun(Integer ndim, const double x[], double fv[],
                            Integer m, Nag_Comm *comm)
{
  Integer i, index, j;
  double sum;

  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback vecfun, first invocation.)\n");
      comm->user[0] = 0.0;
    }
  for (i = 0; i < m; i++)
    {
      sum = 0.0;
      for (j = 0, index = 0; j < ndim; j++, index += m) sum += x[i + index];
      fv[i] = cos(0.5 + 2.0 * sum - 4.0);
    }
}

static void NAG_CALL vecreg(Integer ndim, const double x[], Integer j,
                            double c[], double d[], Integer m, Nag_Comm *comm)
{
  Integer i;

  if (comm->user[1] == -1.0)
    {
      printf("(User-supplied callback vecreg, first invocation.)\n");
      comm->user[1] = 0.0;
    }
  for (i = 0; i < m; i++)
    {
      c[i] = 0.0;
      d[i] = 1.0;
    }
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_quad_md_numth_vec (d01gdc) Example Program Results
(User-supplied callback vecreg, first invocation.)
(User-supplied callback vecfun, first invocation.)
```

```
Result =        0.43999, standard error =   1.89e-06
```