NAG Library Routine Document

D01GDF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

D01GDF calculates an approximation to a definite integral in up to 20 dimensions, using the Korobov–Conroy number theoretic method. This routine is designed to be particularly efficient on vector processors.

2 Specification

```
SUBROUTINE D01GDF (NDIM, VECFUN, VECREG, NPTS, VK, NRAND, ITRANS, RES,     &
                   ERR, IFAIL)

INTEGER             NDIM, NPTS, NRAND, ITRANS, IFAIL
REAL (KIND=nag_wp) VK(NDIM), RES, ERR
EXTERNAL            VECFUN, VECREG
```

3 Description

D01GDF 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 routine 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 routine provides built-in sets of optimal coefficients, corresponding to six different values of p. Alternatively, the optimal coefficients may be supplied by you. Routines D01GYF and D01GZF compute the optimal coefficients for the cases where p is a prime number or p is a product of two primes, respectively.

This routine is designed to be particularly efficient on vector processors, although it is very important that you also code VECFUN and VECREG efficiently.

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: $1 \leq \text{NDIM} \leq 20$.

2: VECFUN – SUBROUTINE, supplied by the user. *External Procedure*

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

The specification of VECFUN is:

```
SUBROUTINE VECFUN (NDIM, X, FV, M)
INTEGER             NDIM, M
REAL (KIND=nag_wp) X(M,NDIM), FV(M)
```

1: NDIM – INTEGER *Input*

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

2: X(M, NDIM) – REAL (KIND=nag_wp) array *Input*

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

3: FV(M) – REAL (KIND=nag_wp) array *Output*

On exit: FV(i) must contain the value of the integrand of the ith point, i.e., FV$(i) = f(\text{X}(i, 1), \text{X}(i, 2), \ldots, \text{X}(i, \text{NDIM}))$, for $i = 1, 2, \ldots, \text{M}$.

4: M – INTEGER *Input*

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

VECFUN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D01GDF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

3: VECREG – SUBROUTINE, supplied by the user. *External Procedure*

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

The specification of VECREG is:

```
SUBROUTINE VECREG (NDIM, X, J, C, D, M)
INTEGER            NDIM, J, M
REAL (KIND=nag_wp) X(M,NDIM), C(M), D(M)
```

1: NDIM – INTEGER *Input*

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

2: X(M, NDIM) – REAL (KIND=nag_wp) array *Input*

On entry: for $i = 1, 2, \ldots, m$, X$(i, 1)$, X$(i, 2), \ldots,$ 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) – REAL (KIND=nag_wp) array *Output*

On exit: C(i) must be set to the lower limit of the range for X(i, j), for $i = 1, 2, \ldots, m$.

5: D(M) – REAL (KIND=nag_wp) array *Output*

On exit: D(i) must be set to the upper limit of the range for 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.

VECREG must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D01GDF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

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) – REAL (KIND=nag_wp) array *Input/Output*

On entry: if NPTS > 6, VK must contain the n optimal coefficients (which may be calculated using D01GYF or D01GZF).

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 routine 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: ITRANS – INTEGER *Input*

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

ITRANS $= 0$
: The transformation is to be used.

ITRANS $\neq 0$
: 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: ITRANS $= 0$.

8: RES – REAL (KIND=nag_wp) *Output*

On exit: the approximation to the integral I.

9: ERR – REAL (KIND=nag_wp) *Output*

On exit: the standard error as computed from NRAND sample values. If NRAND $= 1$, then ERR contains zero.

10: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this argument, the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, NDIM < 1,
or NDIM > 20.

IFAIL $= 2$

On entry, NPTS < 1.

IFAIL $= 3$

On entry, NRAND < 1.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

If NRAND > 1, an estimate of the absolute standard error is given by the value, on exit, of ERR.

8 Parallelism and Performance

D01GDF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

D01GDF performs the same computation as D01GCF. However, the interface has been modified so that it can perform more efficiently on machines with vector processing capabilities. In particular, 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 subroutines, which must be supplied by you, may account for a significant part of the total computation time. For this reason it is vital that you consider the possibilities for vectorization in the code supplied for these two subroutines.

The time taken will be approximately proportional to 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 D01GDF by calls to G05SAF. 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

```
!   D01GDF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d01gdfe_mod

!     D01GDF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                           :: vecfun, vecreg
!     .. Parameters ..
      Integer, Parameter, Public       :: ndim = 4, nout = 6
    Contains
      Subroutine vecfun(ndim,x,fv,m)

!       .. Scalar Arguments ..
        Integer, Intent (In)           :: m, ndim
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: fv(m)
        Real (Kind=nag_wp), Intent (In) :: x(m,ndim)
!       .. Local Scalars ..
        Integer                        :: i
!       .. Intrinsic Procedures ..
        Intrinsic                      :: cos, real, sum
!       .. Executable Statements ..
        Do i = 1, m
          fv(i) = cos(0.5E0_nag_wp+2.0E0_nag_wp*sum(x(i,                    &
            1:ndim))-real(ndim,kind=nag_wp))
        End Do

        Return

      End Subroutine vecfun
      Subroutine vecreg(ndim,x,j,c,d,m)

!       .. Scalar Arguments ..
        Integer, Intent (In)           :: j, m, ndim
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: c(m), d(m)
        Real (Kind=nag_wp), Intent (In) :: x(m,ndim)
!       .. Executable Statements ..
        c(1:m) = 0.0E0_nag_wp
        d(1:m) = 1.0E0_nag_wp

        Return

      End Subroutine vecreg
    End Module d01gdfe_mod
    Program d01gdfe
```

```
!     D01GDF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d01gdf, nag_wp
      Use d01gdfe_mod, Only: ndim, nout, vecfun, vecreg
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)              :: err, res
      Integer                         :: ifail, itrans, npts, nrand
!     .. Local Arrays ..
      Real (Kind=nag_wp)              :: vk(ndim)
!     .. Executable Statements ..
      Write (nout,*) 'D01GDF Example Program Results'

      npts = 2
      itrans = 0
      nrand = 4

      ifail = 0
      Call d01gdf(ndim,vecfun,vecreg,npts,vk,nrand,itrans,res,err,ifail)

      Write (nout,*)
      Write (nout,99999) 'Result = ', res, ', standard error = ', err

99999 Format (1X,A,F13.5,A,E10.2)
    End Program d01gdfe
```

10.2 Program Data

None.

10.3 Program Results

```
 D01GDF Example Program Results

 Result =       0.43999, standard error =   0.19E-05
```