Module 11.3: nag_quad_md
Multi-dimensional Integrals

nag_quad_md provides procedures for computing the value of multi-dimensional definite integrals.

Contents

Introduction

The procedures in this module are designed to estimate the value of a multi-dimensional definite integral of the form:

$$\int_{R_n} f(x_1, x_2, \ldots, x_n)\, dx_n \ldots dx_2 dx_1,$$

where $f(x_1, x_2, \ldots, x_n)$ is a function defined by the user, and R_n is the n-rectangle defined by

$$a_i \leq x_i \leq b_i, \quad i = 1, 2, \ldots, n,$$

where a_i and b_i are constants.

The more general case where a_i and b_i are functions of x_j $(j < i)$ may be dealt with by transforming the region to the rectangular form (see page 226 of Davis and Rabinowitz [1]).

The module also contains a procedure which computes the value of a two-dimensional integral of the form:

$$\int_a^b \int_{\phi_1(y)}^{\phi_2(y)} f(x, y)\, dxdy.$$

where $f(x, y)$ is not badly behaved.

Procedure: nag_quad_md_rect

1 Description

nag_quad_md_rect attempts to compute an approximation to a multi-dimensional integral (up to 15 dimensions) over a hyper-rectangle:

$$I = \int_{a_1}^{b_1} \int_{a_2}^{b_2} \cdots \int_{a_n}^{b_n} f(x_1, x_2, \ldots, x_n)\, dx_n \ \ldots \ dx_2 dx_1,$$

where the limits a_i and b_i, for $i = 1, 2, \ldots, n$ are constant. The procedure returns an estimate of the above integral, and optionally an estimate of the relative error.

This procedure requires a user-supplied function to evaluate the integrand at a given point.

2 Usage

```
USE nag_quad_md
CALL nag_quad_md_rect(f, a, b, result  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

$2 \leq n \leq 15$ — the number of dimensions of the integral

3.1 Mandatory Arguments

f — function

f must return the value of the integrand f at a given point.

```
function f(x)

real(kind=wp), intent(in) ::  x(:)
```

Shape: x has shape (n).

Input: the co-ordinates of the point at which the integrand f must be evaluated.

```
real(kind=wp) ::  f
```

Result: f must contain the value of f at the point with co-ordinates x(i), for $i = 1, 2, \ldots, n$.

a(n) — real(kind=*wp*), intent(in)

b(n) — real(kind=*wp*), intent(in)

Input: the lower and upper limits of integration, a_i and b_i respectively, for $i = 1, 2, \ldots, n$.

result — real(kind=*wp*), intent(out)

Output: the best approximation obtained to the integral I.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

rel_acc — real(kind=*wp*), intent(in), optional

Input: the relative accuracy required. When the solution is zero, or very small, relative accuracy may not be achievable, but you may still set `rel_acc` to a reasonable value and check for the error exit `error%code` = 201.

Default: `rel_acc` = (`EPSILON(1.0_wp)`)$^{1/4}$.

Constraints: `rel_acc` > 0.0.

rel_err — real(kind=*wp*), intent(out), optional

Output: the estimated relative error in `result`.

max_fun_eval — integer, intent(in), optional

Input: maximum number of integrand evaluations to be allowed.

Default: `max_fun_eval` = $200(2^n + 2n^2 + 2n + 1)$.

Constraints: `max_fun_eval` $\geq 2^n + 2n^2 + 2n + 1$.

num_fun_eval — integer, intent(out), optional

Output: the actual number of integrand evaluations.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	The maximum number of function evaluations allowed has been reached. `max_fun_eval` was too small to obtain the specified relative accuracy. `result` and `rel_err` contain estimates of the integral and relative error respectively. However, `rel_err` will be greater than `rel_acc`.

202 Failure due to insufficient internal workspace.

The amount of storage allocated to internal workspace is too small. `result` and `rel_err` contain estimates of the integral and relative error respectively. However, `rel_err` will be greater than `rel_acc`. The size of this workspace is proportional to the value of `max_fun_eval`. Hence, an increase in the value of `max_fun_eval` will result in the allocation of more storage for this internal workspace.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Algorithmic Detail

This procedure is based on the HALF procedure developed by Van Dooren and De Ridder [3]. However, it uses a different basic rule, described by Genz and Malik [2].

The procedure 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 co-ordinate axis are evaluated, and the subregion is marked for possible future subdivision in half along that co-ordinate axis which has the largest absolute fourth difference.

If the estimated errors, totalled over the subregions, exceed the requested relative error, further subdivision is necessary, and is performed on the subregion with the largest estimated error, that subregion being halved along the appropriate co-ordinate axis.

The procedure will fail if the requested relative error level has not been attained by the time `max_fun_eval` calls to `f` have been made, or if the amount of internal working storage is insufficient. The amount of working storage allocated internally by the procedure is sufficient for most problems. However, if this amount is exhausted in the course of execution, the procedure switches to a less efficient mode of operation; the procedure will report an error only if this mode also breaks down. In this case you are advised to increase the value of `max_fun_eval`.

In general, convergence is fast for well behaved integrands and highly accurate results can often be obtained for n between 2 and 5. The procedure will usually work when the integrand is mildly singular.

6.2 Accuracy

A relative error estimate is available through the optional argument `rel_err`.

6.3 Timing

Execution time will usually be dominated by the time taken to evaluate the integrand `f`, and hence the maximum time required will be proportional to `max_fun_eval`.

Procedure: nag_quad_md_rect_mintg

1 Description

nag_quad_md_rect_mintg computes approximations to the integrals of a vector of similar functions:

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2}\cdots\int_{a_n}^{b_n}(f_1,f_2,\ldots,f_m)\,dx_n\ \ldots\ dx_2dx_1,$$

where $f_i = f_i(x_1,x_2,\ldots,x_n)$, for $i = 1,2,\ldots,m$; that is each function is defined over the same multi-dimensional hyper-rectangle region.

This procedure requires a vector valued user-supplied function to evaluate the integrand at a given point.

2 Usage

```
USE nag_quad_md
CALL nag_quad_md_rect_mintg(f, a, b, result  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$n \geq 1$ — the number of dimensions of the integral
$m \geq 1$ — the number of integrands

3.1 Mandatory Arguments

f — function

f must return the values of the integrands f_i, for $i = 1,2,\ldots,m$, at a given point.

```
function f(x,m)

real(kind=wp), intent(in) ::  x(:)
```
Shape: x has shape (n).
Input: the co-ordinates of the point at which the integrands must be evaluated.

```
integer, intent(in) ::  m
```
Input: the number of integrands, m.

```
real(kind=wp) ::  f(m)
```
Result: the values of the m integrands at a point with co-ordinates x(i), for $i = 1,2,\ldots,n$.

a(n) — real(kind=*wp*), intent(in)
b(n) — real(kind=*wp*), intent(in)

Input: the lower and upper limits of integration, a_i and b_i, for $i = 1,2,\ldots,n$, respectively.

result(*m*) — real(kind=*wp*), intent(out)

Output: `result(`i`)` specifies the best estimated result obtained from the ith integral, for $i = 1, 2, \ldots, m$.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

abs_acc — real(kind=*wp*), intent(in), optional

Input: the absolute accuracy required.

Note: both `abs_acc` and `rel_acc` cannot be zero.

Default: `abs_acc` = `SQRT(EPSILON(1.0_`*wp*`))`.

Constraints: `abs_acc` ≥ 0.0.

rel_acc — real(kind=*wp*), intent(in), optional

Input: the relative accuracy required.

Note: both `rel_acc` and `abs_acc` cannot be zero.

Default: `rel_acc` $= 10^{-4}$.

Constraints: `rel_acc` ≥ 0.0.

abs_err(*m*) — real(kind=*wp*), intent(out), optional

Output: `abs_err(`i`)` is the estimated absolute accuracy of `result(`i`)`, for $i = 1, 2, \ldots, m$.

min_fun_eval — integer, intent(in), optional

Input: minimum number of calls to `f`.

Note: `min_fun_eval` has no effect on the computation when `continue_call` = `.true.`.

Default: `min_fun_eval` $= 0$.

Constraints: `min_fun_eval` ≥ 0.

max_fun_eval — integer, intent(in), optional

Input: maximum number of calls to `f`.

Default:

`max_fun_eval` $= 50000$, if $n < 11$,

`max_fun_eval` $= 200n^3$, if $n \geq 11$.

Constraints: `max_fun_eval` $\geq \max($`min_fun_eval`$,r)$ where

$r = 2^n + 2n^2 + 2n + 1$, if $n < 11$,

$r = 1 + n(4n^2 - 6n + 14)/3$, if $n \geq 11$.

continue_call — logical, intent(in), optional

Input: if `continue_call` = `.true.`, the procedure continues the calculation started in a previous call with the same integrands and integration limits; no arguments other than `max_fun_eval`, `abs_acc`, `rel_acc` or `error` may be changed between the calls. Otherwise, no action is taken between the calls.

Default: `continue_call` = `.false.`.

Constraints: if `continue_call` is present as an argument, then the array `subrgn_info` must also be present as an argument.

num_fun_eval — integer, intent(out), optional

Output: the actual number of function calls used. In the continuation case this is the number of new calls to `f` made on the current call to this procedure.

subrgn_info(k) — real(kind=*wp*), intent(inout), optional

Input: if `continue_call = .true.`, `subrgn_info` should be present and must be unchanged from the previous call to this procedure.

Output: `subrgn_info` contains information about the current subdivision which could be used in a continuation call.

Default: $k = 6n + 9m + (n + m + 2)(1 + p/r)$. See below.

Recommended Value: $k \geq 6n + 9m + (n + m + 2)(1 + p/r)$, where $p =$ `max_fun_eval` and r is defined under `max_fun_eval`. If k is significantly smaller than this, the procedure will not work as efficiently and may even fail.

Constraints: $k \geq 8n + 11m + 3$.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	The maximum number of function evaluations allowed has been reached. `max_fun_eval` was too small to obtain the specified relative accuracy. The arrays `result` and `abs_err` contain estimates of the integrals and the errors respectively.
202	The amount of storage for the workspace has been exceeded. If the argument `subrgn_info` is present then its size is too small for the procedure to continue. The arrays `result` and and `abs_err` contain current estimates for the integrals and the errors respectively. However, if the argument `subrgn_info` is not present then the amount of storage for this argument has been exceeded. In this case, supply `subrgn_info` as an argument and increase its size.
203	Insufficient value of `max_fun_eval` on a continuation call. On a continuation call, `max_fun_eval` was set too small to make any progress. Increase `max_fun_eval` before calling this procedure again.

5 Examples of Usage

Complete examples of the use of this procedure appear in Examples 2 and 3 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The procedure uses a globally adaptive method based on the algorithm described by Van Dooren and De Ridder [3], and Genz and Malik [2]. Upon entry, unless `min_fun_eval` has been set to a value equal to zero, the procedure divides the integration region into a number of subregions with randomly selected volumes. Inside each subregion the integrals and their errors are estimated. The initial number of subregions is chosen to be as large as possible without using more than `min_fun_eval` calls to `f`. The results are stored in a partially ordered list (a heap). The procedure then proceeds in stages. At each stage the subregion with the largest error (measured using the maximum norm) is halved along the co-ordinate axis where the integrands have largest absolute fourth differences. The basic rule is applied to each half of this subregion and the results are stored in the list. The results from the two halves are used to update the global integral and error estimates (`result` and `abs_err`) and the procedure continues unless

$$\|\texttt{result}\| \leq \max(\texttt{abs\_err}, \|\texttt{result}\| \times \texttt{rel\_acc})$$

where $\|.\|$ is the maximum norm, or further subdivision would use more than `max_fun_eval` calls to `f`. If at some stage there is insufficient working storage to keep the results for the next subdivision, the procedure switches to a less efficient mode; only if this mode of operation breaks down is insufficient storage reported.

However, for some integrands, particularly those that are poorly behaved in a small part of the integration region, this procedure may terminate prematurely with values of `abs_err` that are significantly smaller than the actual absolute errors. This behaviour should be suspected if the returned value of `num_fun_eval` is small relative to the expected difficulty of the integrals. When this occurs this procedure should be called again, but with an entry value of `min_fun_eval` $\geq 2r$ (see specification of `max_fun_eval`), and the results compared with those from the previous call.

If the procedure is called with `min_fun_eval` $\geq 2r$, the exact values of `result` and `abs_err` on return will depend (within statistical limits) on the sequence of random numbers generated internally within this procedure by calls to `nag_rand_uniform`. Separate runs will produce identical answers unless the part of the program executed prior to calling `nag_quad_md_rect_mintg` also calls (directly or indirectly) procedures from Chapter 21, and, in addition, the series of such calls differs between runs.

Because of moderate instability in the application of the basic integration rule, approximately the last $1 + \log_{10}(n^3)$ decimal digits may be inaccurate when using this procedure for large values of n.

6.2 Accuracy

An absolute error estimate for each integrand is available through the optional array argument `abs_err`. The procedure exits successfully if

$$\max_i(\texttt{abs\_err}(i)) \leq \max(\texttt{abs\_acc}, \texttt{rel\_acc} \times \max_i \mid \texttt{abs\_err}(i) \mid).$$

6.3 Timing

Execution time will usually be dominated by the time to evaluate the integrands, and hence the maximum time required will be proportional to `max_fun_eval`.

Procedure: nag_quad_2d

1 Description

nag_quad_2d evaluates an approximation to the double integral

$$I = \int_a^b \int_{\phi_1(y)}^{\phi_2(y)} f(x,y)\, dxdy$$

to a specified absolute accuracy by repeated applications of the method described by Patterson [5] and Patterson [6], where the limits a and b are constants and $\phi_1(y)$ and $\phi_2(y)$ are functions of the variable y.

This procedure requires a user-supplied function to evaluate the integrand at a given point. It also requires two user-supplied functions to evaluate the limits of the inner intergration.

2 Usage

```
USE nag_quad_md
CALL nag_quad_2d(f, phi1, phi2, a, b, result  [, optional arguments])
```

3 Arguments

3.1 Mandatory Arguments

f — function

f must return the value of the integrand f at a given two-dimensional point.

```
function f(x,y)

real(kind=wp), intent(in) ::  x
real(kind=wp), intent(in) ::  y
```
Input: the co-ordinates of the point (x, y) at which the integrand must be evaluated.

```
real(kind=wp) ::  f
```
Result: f must contain the value of f at the point with co-ordinates x and y.

phi1 — function

phi1 must return the lower limit of the inner integral for a given value of y.

```
function phi1(y)

real(kind=wp), intent(in) ::  y
```
Input: the value of y for which the lower limit must be evaluated.

```
real(kind=wp) ::  phi1
```
Result: phi1 must contain the value of ϕ_1 at the point y.

phi2 — function

phi2 must return the upper limit of the inner integral for a given value of y.

```
function phi2(y)

real(kind=wp), intent(in) ::  y
```
Input: the value of y for which the upper limit must be evaluated.

```
real(kind=wp) ::  phi2
```
Result: phi2 must contain the value of ϕ_2 at the point y.

a — real(kind=*wp*), intent(in)

b — real(kind=*wp*), intent(in)

Input: the upper and the lower limits of the integral, a and b. It is not necessary that $a < b$.

result — real(kind=*wp*), intent(out)

Output: the value of the computed integral.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

abs_acc — real(kind=*wp*), intent(in), optional

Input: the absolute accuracy requested for the outer integral.

Default: abs_acc = 10^{-4}.

abs_acc_inner — real(kind=*wp*), intent(in), optional

Input: the absolute accuracy requested for the inner integral.

Note: this argument is provided for expert users. If you are not familiar with the product integration rule, you are advised to use the default value as in the vast majority of cases this has proved to be adequate for the overall result of the double integral.

Default: abs_acc_inner = 0.1×abs_acc.

Constraints: abs_acc_inner $\leq$ abs_acc.
If abs_acc_inner is present then abs_acc must also be present.

num_fun_eval — integer, intent(out), optional

Output: the total number of integrand evaluations.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	Invalid absence of an optional argument. This indicates that the optional argument `abs_acc_inner` is present when the optional argument `abs_acc` is absent.

Failures (error%level = 2):

error%code	Description
201	Failure due to convergence not being achieved in the outer integral This indicates that 255 points have been used in the outer integral and convergence has not been obtained. All the inner integrals have, however, converged. In this case `result` may still contain an approximate estimate of the integral.
202	Failure due to convergence not being achieved in outer integral and some inner integrals This indicates that both the outer integral and some inner integrals have failed to converge. `result` may still contain an approximate estimate of the integral, but its reliability will decrease as the number on non-convergent inner integrals increases. An error message reports the precise number of these inner integrals.

Warnings (error%level = 1):

error%code	Description
101	Convergence was not achieved in some inner integrals This indicates that while the outer integral has converged some inner integrals failed to converge with the use of 255 points. In this case `result` may still contain an approximate estimate of the integral, but its reliability will decrease as the the number of non-convergent integrals increases. The warning reports the precise number of the inner integrals that failed to converge.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 4 of this module document.

6 Further Comments

6.1 Algorithmic Detail

This procedure evaluates a definite integral of the form

$$I = \int_a^b \int_{\phi_1(y)}^{\phi_2(y)} f(x,y)dxdy$$

where a and b are constants and $\phi_1(y)$ and $\phi_2(y)$ are functions of the variable y.

The integral is evaluated by expressing it as

$$I = \int_a^b F(y)dy, \text{ where } F(y) = \int_{\phi_1(y)}^{\phi_2(y)} f(x,y)dx.$$

Both the outer integral I and the inner integrals $F(y)$ are evaluated by the method, described by Patterson [5] and Patterson [6], of the optimum addition of points to Gauss quadrature formulae.

This method uses a family of interlacing common point formulae. Beginning with the 3-point Gauss rule, formulae using 7, 15, 31, 63, 127 and finally 255 points are derived. Each new formula contains all the abscissae of the earlier formulae so that no function evaluations are wasted. Each integral is evaluated by applying these formulae successively until two results are obtained which differ by less than the specified absolute accuracy.

With Patterson's method accidental convergence may occasionally occur, when two estimates of an integral agree to within the requested accuracy, but both estimates differ considerably from the true result. This could occur in either the outer integral or in one or more of the inner integrals.

If it occurs in the outer integral then apparent convergence is likely to be obtained with considerably fewer integrand evaluations than may be expected. If it occurs in an inner integral, the incorrect value could make the function $F(y)$ appear to be badly behaved, in which case a very large number of abscissae may be needed for the overall evaluation of the integral. Thus both unexpectedly small and unexpectedly large numbers of integrand evaluations should be considered as indicating possible trouble. If accidental convergence is suspected, the integral may be recomputed, requesting better accuracy; if the new request is more stringent than the degree of accidental agreement (which is of course unknown), improved results should be obtained. This is only possible when the accidental agreement is not better than machine accuracy.

The procedure is not well suited to non-smooth integrands, i.e., integrands having some kind of analytic discontinuity (such as a discontinuous or infinite partial derivative of some low order) in, on the boundary of, or near, the region of integration.

Warning: such singularities may be induced by incautiously presenting an apparently smooth interval over the positive quadrant of the unit circle, R

$$I = \int_R (x+y)dxdy.$$

This may be presented to this procedure as

$$I = \int_0^1 dy \int_0^{\sqrt{1-y^2}} (x+y)dx = \int_0^1 i\left(\frac{1}{2}(1-y^2) + y\sqrt{1-y^2}\right)dy$$

but here the outer integral has an induced square-root singularity stemming from the way the region has been presented to this procedure. This situation should be avoided by re-casting the problem. For the example given, the use of polar co-ordinates would avoid the difficulty:

$$I = \int_0^1 dr \int_0^{\pi/2} r^2(\cos\upsilon + \sin\upsilon)d\upsilon.$$

6.2 Accuracy

If on exit no failure is reported then the result is most likely correct to the requested accuracy `abs_acc`. Even if a failure is reported on exit, it is still possible that the calculated result could differ from the true value by less than the given accuracy.

Procedure: nag_quad_monte_carlo

1 Description

nag_quad_monte_carlo computes an approximation to a multi-dimensional integral over a hyper-rectangle region, using a Monte-Carlo method. The integral has the form:

$$I = \int_{a_1}^{b_1} \int_{a_2}^{b_2} \cdots \int_{a_n}^{b_n} f(x_1, x_2, \ldots, x_n)\, dx_n \ \ldots\ dx_2 dx_1,$$

where the limits a_i and b_i, for $i = 1, 2, \ldots, n$ are constant. The procedure returns an estimate of the above integral, and optionally an estimate of the relative error.

This procedure is suitable for low accuracy work and requires a user-supplied function to evaluate the integrand at a given point.

2 Usage

```
USE nag_quad_md

CALL nag_quad_monte_carlo(f, a, b, result  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

$n \geq 1$ — the number of dimensions of the integral

3.1 Mandatory Arguments

f — function

f must return the value of the integrand f at a given point.

```
function f(x)

real(kind=wp), intent(in) ::  x(:)
```

Shape: x has shape (n).

Input: the co-ordinates of the point at which the integrand f must be evaluated.

```
real(kind=wp) ::  f
```

Result: f must contain the value of f at the point with co-ordinates x(i), for $i = 1, 2, \ldots,$ SIZE(x).

a(n) — real(kind=*wp*), intent(in)

b(n) — real(kind=*wp*), intent(in)

Input: the lower and upper limits of integration, a_i and b_i respectively, for $i = 1, 2, \ldots, n$.

result — real(kind=*wp*), intent(out)

Output: the best approximation obtained to the integral I.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

rel_acc — real(kind=*wp*), intent(in), optional

Input: the relative accuracy required. When the solution is zero, or very small, relative accuracy may not be achievable, but you may still set `rel_acc` to a reasonable value and check for the error exit `error%code` = 201.

Default: `rel_acc` = MAX$(0.001, (\texttt{EPSILON(1.0_}wp\texttt{)})^{1/5})$.

Constraints: `rel_acc` > 0.0.

rel_err — real(kind=*wp*), intent(out), optional

Output: the estimated relative error in `result`.

max_fun_eval — integer, intent(in), optional

Input: maximum number of integrand evaluations to be allowed.

Default: `max_fun_eval` $= 4000(n+1)$.

Constraints: `max_fun_eval` $\geq 4(n+1)$.

num_fun_eval — integer, intent(out), optional

Output: the actual number of integrand evaluations.

repeat_rand — logical, intent(in), optional

Input: if `repeat_rand` = `.true.`, the initial value of the seed supplied to the random numbers generator used by `nag_quad_monte_carlo` is reset to create a repeatable sequence of random numbers. This will ensure that separate runs will produce identical answers. Otherwise, the seed is calculated from the system clock, resulting in a non-repeatable sequence hence non-identical answers (for more details see Section 6.1).

Default: `repeat_rand` = `.false.`.

continue_call — logical, intent(in), optional

Input: if `continue_call` = `.true.`, the procedure continues the calculation started in a previous call with the same integrands and integration limits; no arguments other than `max_fun_eval`, `abs_acc`, `rel_acc` or `error` may be changed between the calls. Otherwise, no action is taken between the calls.

Default: `continue_call` = `.false.`.

Constraints: if `continue_call` is present as an argument, then the array `subrgn_info` must also be present as an argument.

subrgn_info(k) — real(kind=*wp*), intent(inout), optional

Input: if `continue_call` = `.true.`, `subrgn_info` should be present and must be unchanged from the previous call to this procedure.

Output: `subrgn_info` contains information about the current subdivision which could be used in a continuation call.

Default: $k = n + 3n(\texttt{max\_fun\_eval}/4)^{1/n}$. See below.

Recommended Value: $k \geq n + 3n(\texttt{max\_fun\_eval}/4)^{1/n}$. If k is significantly smaller than this, the procedure will not work as efficiently and may even fail.

Constraints: $k \geq 4n$.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	The maximum number of function evaluations allowed has been reached. max_fun_eval was too small to obtain the specified relative accuracy. result and rel_err contain estimates of the integral and relative error respectively. However, rel_err will be greater than rel_acc.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 5 of this module document.

6 Further Comments

6.1 Algorithmic Detail

nag_quad_monte_carlo uses an adaptive Monte Carlo method based on the algorithm described by Lautrup [4]. It is implemented for integrals of the form:

$$\int_{a_1}^{b_1} \int_{a_2}^{b_2} \dots \int_{a_n}^{b_n} f(x_1, x_2, \dots, x_n)\, dx_n \ \dots\, dx_2\, dx_1.$$

Upon entry, the procedure 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 procedure 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.

For some integrands, particularly those that are poorly behaved in a small part of the integration region, nag_quad_monte_carlo may terminate with a value of rel_err which is significantly smaller than the actual relative error. This should be suspected if the value of the optional argument num_fun_eval is small relative to the expected difficulty of the integral. Where this occurs, nag_quad_monte_carlo should

be called again, but with a higher entry value of `max_fun_eval` (e.g., twice the value used in the previous call) and the results compared with those from the previous call.

The exact values of `result` and `rel_err` on return will depend (within statistical limits) on the sequence of random numbers generated within `nag_quad_monte_carlo` by calls to `nag_rand_uniform`. Separate runs will produce identical answers unless the optional argument `repeat_rand` is used and set to `.false.`.

6.2 Timing

Execution time will for `nag_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 `max_fun_eval`.

Example 1: Evaluation of a Four-dimensional Integral Using nag_quad_md_rect

The four-dimensional integral

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{4x_1 x_3^2 \exp(2x_1 x_3)}{(1+x_2+x_4)^2}\, dx_4\, dx_3\, dx_2\, dx_1$$

is computed using the procedure nag_quad_md_rect.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
MODULE quad_md_ex01_mod

  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Default Accessibility ..
  PUBLIC
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)

CONTAINS

  FUNCTION f(x)

    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Intrinsic Functions ..
    INTRINSIC EXP
    ! .. Array Arguments ..
    REAL (wp), INTENT (IN) :: x(:)
    ! .. Function Return Value ..
    REAL (wp) :: f
    ! .. Executable Statements ..
    f = 4.0_wp*x(1)*x(3)*x(3)*EXP(2.0_wp*x(1)*x(3))/(1.0_wp+x(2)+x(4))**2

  END FUNCTION f

END MODULE quad_md_ex01_mod

PROGRAM nag_quad_md_ex01

  ! Example Program Text for nag_quad_md
  ! NAG f190, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_quad_md, ONLY : nag_quad_md_rect
  USE quad_md_ex01_mod, ONLY : wp, f
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Parameters ..
  INTEGER, PARAMETER :: n = 4
  ! .. Local Scalars ..
  REAL (wp) :: result
  ! .. Local Arrays ..
  REAL (wp) :: a(n), b(n)
```

```
      ! .. Executable Statements ..
      WRITE (nag_std_out,*) 'Example Program Results for nag_quad_md_ex01'

      a = 0.0_wp
      b = 1.0_wp

      CALL nag_quad_md_rect(f,a,b,result)

      WRITE (nag_std_out,'(/,1X,A,F9.5)') &
       'result - approximation to the integral =', result

    END PROGRAM nag_quad_md_ex01
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_quad_md_ex01

 result - approximation to the integral =  0.57536
```

Example 2: Evaluation of Ten Four-dimensional Integrals

The four-dimensional integrals

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 (f_1, f_2, \ldots, f_{10})\, dx_4\, dx_3\, dx_2\, dx_1,$$

where

$$f_j = \ln(x_1 + 2x_2 + 3x_3 + 4x_4)\sin(j + x_1 + 2x_2 + 3x_3 + 4x_4),$$

for $j = 1, 2, \ldots, 10$, are computed using the procedure `nag_quad_md_rect_mintg`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
MODULE quad_md_ex02_mod

  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Default Accessibility ..
  PUBLIC
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)

CONTAINS

  FUNCTION f(x,m)

    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Intrinsic Functions ..
    INTRINSIC LOG, REAL, SIN, SIZE, SUM
    ! .. Scalar Arguments ..
    INTEGER, INTENT (IN) :: m
    ! .. Array Arguments ..
    REAL (wp), INTENT (IN) :: x(:)
    ! .. Function Return Value ..
    REAL (wp) :: f(m)
    ! .. Local Scalars ..
    INTEGER :: j
    REAL (wp) :: s
    ! .. Executable Statements ..
    s = SUM((/(REAL(j,kind=wp),j=1,SIZE(x))/)*x)
    f = LOG(s)*(/ (SIN(REAL(j,kind=wp)+s),j=1,m) /)

  END FUNCTION f

END MODULE quad_md_ex02_mod

PROGRAM nag_quad_md_ex02

  ! Example Program Text for nag_quad_md
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_quad_md, ONLY : nag_quad_md_rect_mintg
  USE quad_md_ex02_mod, ONLY : wp, f
```

```
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Parameters ..
  INTEGER, PARAMETER :: m = 10, n = 4
  ! .. Local Scalars ..
  INTEGER :: i
  ! .. Local Arrays ..
  REAL (wp) :: a(n), b(n), result(m)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_quad_md_ex02'

  a = 0.0_wp
  b = 1.0_wp

  CALL nag_quad_md_rect_mintg(f,a,b,result)

  WRITE (nag_std_out,'(/,1X,A)') '   i       integral'
  DO i = 1, m
    WRITE (nag_std_out,'(1X,I4,F14.4)') i, result(i)
  END DO

END PROGRAM nag_quad_md_ex02
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_quad_md_ex02

    i       integral
    1        0.0383
    2        0.4012
    3        0.3952
    4        0.0258
    5       -0.3672
    6       -0.4227
    7       -0.0895
    8        0.3260
    9        0.4417
   10        0.1514
```

Example 3: Illustration of the Continuation Facility in nag_quad_md_rect_mintg

The four-dimensional integrals

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 (f_1, f_2, \ldots, f_{10})\, dx_4\, dx_3\, dx_2\, dx_1,$$

where

$$f_j = \ln(x_1 + 2x_2 + 3x_3 + 4x_4)\sin(j + x_1 + 2x_2 + 3x_3 + 4x_4),$$

for $j = 1, 2, \ldots, 10$, are computed using the procedure **nag_quad_md_rect_mintg**. This example program is intended to illustrate the continuation facility provided by **nag_quad_md_rect_mintg**: the procedure exits with **error%code** = 201 (printing an explanatory error message) and is re-entered with **max_fun_eval** reset to a larger value. The program can be used with different numbers of integrands and dimensions. The program also uses the module **nag_error_handling** (1.2) to control the error state.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
MODULE quad_md_ex03_mod

  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Default Accessibility ..
  PUBLIC
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)

CONTAINS

  FUNCTION f(x,m)

    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Intrinsic Functions ..
    INTRINSIC LOG, REAL, SIN, SIZE, SUM
    ! .. Scalar Arguments ..
    INTEGER, INTENT (IN) :: m
    ! .. Array Arguments ..
    REAL (wp), INTENT (IN) :: x(:)
    ! .. Function Return Value ..
    REAL (wp) :: f(m)
    ! .. Local Scalars ..
    INTEGER :: j
    REAL (wp) :: s
    ! .. Executable Statements ..
    s = SUM((/(REAL(j,kind=wp),j=1,SIZE(x))/)*x)
    f = LOG(s)*(/ (SIN(REAL(j,kind=wp)+s),j=1,m) /)

  END FUNCTION f

END MODULE quad_md_ex03_mod

PROGRAM nag_quad_md_ex03

  ! Example Program Text for nag_quad_md
```

```
      ! NAG fl90, Release 4. NAG Copyright 2000.

      ! .. Use Statements ..
      USE nag_examples_io, ONLY : nag_std_out
      USE nag_quad_md, ONLY : nag_quad_md_rect_mintg, nag_set_error, nag_error
      USE quad_md_ex03_mod, ONLY : wp, f
      ! .. Implicit None Statement ..
      IMPLICIT NONE
      ! .. Parameters ..
      INTEGER, PARAMETER :: m = 10, n = 4
      ! .. Local Scalars ..
      INTEGER :: i, len_info, max_fun_eval, num_fun_eval, r
      REAL (wp) :: abs_acc, rel_acc
      LOGICAL :: continue_call
      TYPE (nag_error) :: error
      ! .. Local Arrays ..
      REAL (wp) :: a(n), abs_err(m), b(n), result(m)
      REAL (wp), ALLOCATABLE :: subrgn_info(:)
      ! .. Executable Statements ..
      WRITE (nag_std_out,*) 'Example Program Results for nag_quad_md_ex03'
      WRITE (nag_std_out,*)

      r = 2**n + 2*n*n + 2*n + 1
      max_fun_eval = 57
      len_info = 6*n + 9*m + (n+m+2)*(1+max_fun_eval/r)

      ALLOCATE (subrgn_info(len_info)) ! Allocate storage

      a = 0.0_wp
      b = 1.0_wp
      abs_acc = 0.0_wp
      rel_acc = 1.0E-3_wp
      max_fun_eval = 57
      continue_call = .FALSE.

inner: DO

        CALL nag_set_error(error,halt_level=3)

        CALL nag_quad_md_rect_mintg(f,a,b,result,abs_acc=abs_acc, &
         rel_acc=rel_acc,abs_err=abs_err,max_fun_eval=max_fun_eval, &
         num_fun_eval=num_fun_eval,continue_call=continue_call, &
         subrgn_info=subrgn_info,error=error)

        IF (error%code==201 .OR. error%code==203) THEN

          WRITE (nag_std_out,'(/,1X,A,I6)') &
           'Number of function calls in the last call = ', num_fun_eval

          WRITE (nag_std_out,'(1X,A)') &
           '   i      Integral     Estimated error'
          DO i = 1, m
            WRITE (nag_std_out,'(1X,I4,F12.4,E19.4)') i, result(i), abs_err(i)
          END DO
          max_fun_eval = 16*max_fun_eval
          continue_call = .TRUE.
          WRITE (nag_std_out,*)

        ELSE

          WRITE (nag_std_out,'(/,1X,A,I6)') &
           'Number of function calls in the final call = ', num_fun_eval
```

```
          WRITE (nag_std_out,'(1X,A)') &
            '   i      integral     Estimated error'
          DO i = 1, m
            WRITE (nag_std_out,'(1X,I4,F12.4,E19.4)') i, result(i), abs_err(i)
          END DO
          EXIT inner
        END IF
      END DO inner

      DEALLOCATE (subrgn_info)     ! Deallocate storage

    END PROGRAM nag_quad_md_ex03
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_quad_md_ex03

 ****************** Failure reported by NAG Fortran 90 Library *****************
 Procedure nag_quad_md_rect_mintg           Level = 2  Code = 201
 Failure due to insufficient max_fun_eval:
 The allowed maximum number of function  evaluations has been
 reached, without achieving the required accuracy.
 (See argument max_fun_eval.)
 ***************************** Execution continued ******************************

 Number of function calls in the last call =     57
    i      Integral     Estimated error
    1      0.0422         0.8552E-02
    2      0.3998         0.3809E-02
    3      0.3898         0.1267E-01
    4      0.0214         0.9880E-02
    5     -0.3666         0.1991E-02
    6     -0.4176         0.1203E-01
    7     -0.0846         0.1101E-01
    8      0.3261         0.1339E-03
    9      0.4371         0.1116E-01
   10      0.1461         0.1192E-01

 ****************** Failure reported by NAG Fortran 90 Library *****************
 Procedure nag_quad_md_rect_mintg           Level = 2  Code = 201
 Failure due to insufficient max_fun_eval:
 The allowed maximum number of function  evaluations has been
 reached, without achieving the required accuracy.
 (See argument max_fun_eval.)
 ***************************** Execution continued ******************************

 Number of function calls in the last call =    798
    i      Integral     Estimated error
    1      0.0384         0.5837E-03
    2      0.4012         0.5609E-03
    3      0.3952         0.5853E-03
    4      0.0258         0.5928E-03
    5     -0.3673         0.5715E-03
    6     -0.4227         0.5832E-03
    7     -0.0895         0.6107E-03
    8      0.3260         0.5797E-03
    9      0.4417         0.5694E-03
```

```
  10      0.1514         0.6164E-03


 Number of function calls in the final call =    912
    i      integral     Estimated error
    1      0.0384         0.3536E-03
    2      0.4012         0.3164E-03
    3      0.3952         0.3062E-03
    4      0.0258         0.3490E-03
    5     -0.3672         0.3284E-03
    6     -0.4227         0.3136E-03
    7     -0.0895         0.3487E-03
    8      0.3260         0.3408E-03
    9      0.4417         0.3148E-03
   10      0.1514         0.3414E-03
```

Example 4: Evaluation of a Two-dimensional Integral

The integral

$$\int_0^1 \int_0^{\sqrt{1-y^2}} (x+y)\,dx\,dy,$$

is evaluated using `nag_quad_2d`. We then re-cast the problem (see Section 6.2) as:

$$\int_0^1 \int_0^{\pi/2} r^2(\cos v + \sin v)\,dv\,dr.$$

and evaluate the re-cast integral using `nag_quad_2d`.
Note the difference in the number of function evaluations.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
MODULE quad_md_ex04_mod
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Default Accessibility ..
  PUBLIC
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  REAL (wp) :: pi

CONTAINS
  FUNCTION fa(x,y)
    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Scalar Arguments ..
    REAL (wp), INTENT (IN) :: x, y
    ! .. Function Return Value ..
    REAL (wp) :: fa
    ! .. Executable Statements ..
    fa = x + y
  END FUNCTION fa

  FUNCTION phi1a(y)
    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Scalar Arguments ..
    REAL (wp), INTENT (IN) :: y
    ! .. Function Return Value ..
    REAL (wp) :: phi1a
    ! .. Executable Statements ..
    phi1a = 0.0_wp
  END FUNCTION phi1a

  FUNCTION phi2a(y)
    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Intrinsic Functions ..
    INTRINSIC SQRT
    ! .. Scalar Arguments ..
    REAL (wp), INTENT (IN) :: y
```

```
    ! .. Function Return Value ..
    REAL (wp) :: phi2a
    ! .. Executable Statements ..
    phi2a = SQRT(1.0_wp-y*y)
  END FUNCTION phi2a
  ! Second formulation

  FUNCTION fb(x,y)
    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Intrinsic Functions ..
    INTRINSIC COS, SIN
    ! .. Scalar Arguments ..
    REAL (wp), INTENT (IN) :: x, y
    ! .. Function Return Value ..
    REAL (wp) :: fb
    ! .. Executable Statements ..
    fb = (y*y)*(COS(x)+SIN(x))
  END FUNCTION fb

  FUNCTION phi1b(y)
    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Scalar Arguments ..
    REAL (wp), INTENT (IN) :: y
    ! .. Function Return Value ..
    REAL (wp) :: phi1b
    ! .. Executable Statements ..
    phi1b = 0.0_wp
  END FUNCTION phi1b

  FUNCTION phi2b(y)
    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Scalar Arguments ..
    REAL (wp), INTENT (IN) :: y
    ! .. Function Return Value ..
    REAL (wp) :: phi2b
    ! .. Executable Statements ..
    phi2b = 0.5_wp*pi
  END FUNCTION phi2b

END MODULE quad_md_ex04_mod

PROGRAM nag_quad_md_ex04

  ! Example Program Text for nag_quad_md
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_math_constants, ONLY : nag_pi
  USE nag_quad_md, ONLY : nag_quad_2d
  USE quad_md_ex04_mod, ONLY : wp, fa, fb, phi1a, phi2a, phi1b, phi2b, pi
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Local Scalars ..
  INTEGER :: nfun
  REAL (wp) :: a, b, result
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_quad_md_ex04'
  pi = nag_pi(0.0_wp)
```

```
    a = 0.0_wp
    b = 1.0_wp

    CALL nag_quad_2d(fa,phi1a,phi2a,a,b,result,num_fun_eval=nfun)

    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) 'Results for first formulation:'
    WRITE (nag_std_out,'(5X,A,F11.6)') 'The integral is:', result
    WRITE (nag_std_out,'(5X,A,I5)') 'The number of function evaluations:', &
     nfun
    ! Second formulation of the integral

    CALL nag_quad_2d(fb,phi1b,phi2b,a,b,result,num_fun_eval=nfun)

    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) 'Results for second formulation:'
    WRITE (nag_std_out,'(5X,A,F11.6)') 'The integral is:', result
    WRITE (nag_std_out,'(5X,A,I5)') 'The number of function evaluations:', &
     nfun
  END PROGRAM nag_quad_md_ex04
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_quad_md_ex04

 Results for first formulation:
     The integral is:   0.666667
     The number of function evaluations:   93

 Results for second formulation:
     The integral is:   0.666667
     The number of function evaluations:   65
```

Example 5: Evaluation of a Four-dimensional Integral Using nag_quad_monte_carlo

The four-dimensional integral

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{4x_1x_3^2\exp(2x_1x_3)}{(1+x_2+x_4)^2}\,dx_4\,dx_3\,dx_2\,dx_1$$

is computed using the procedure nag_quad_monte_carlo.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
MODULE quad_md_ex05_mod

  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Default Accessibility ..
  PUBLIC
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)

CONTAINS

  FUNCTION f(x)

    ! .. Implicit None Statement ..
    IMPLICIT NONE
    ! .. Intrinsic Functions ..
    INTRINSIC EXP
    ! .. Array Arguments ..
    REAL (wp), INTENT (IN) :: x(:)
    ! .. Function Return Value ..
    REAL (wp) :: f
    ! .. Executable Statements ..
    f = 4.0_wp*x(1)*x(3)*x(3)*EXP(2.0_wp*x(1)*x(3))/(1.0_wp+x(2)+x(4))**2

  END FUNCTION f

END MODULE quad_md_ex05_mod

PROGRAM nag_quad_md_ex05

  ! Example Program Text for nag_quad_md
  ! NAG f190, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_quad_md, ONLY : nag_quad_monte_carlo
  USE quad_md_ex05_mod, ONLY : wp, f
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC EPSILON, MAX
  ! .. Parameters ..
  INTEGER, PARAMETER :: n = 4
  ! .. Local Scalars ..
  INTEGER :: i, num_fun_eval
```

```
      REAL (wp) :: result
      ! .. Local Arrays ..
      REAL (wp) :: a(n), b(n), rel_acc(2)
      ! .. Executable Statements ..
      WRITE (nag_std_out,*) 'Example Program Results for nag_quad_md_ex05'

      a = 0.0_wp
      b = 1.0_wp
      rel_acc(1) = 0.01_wp
      rel_acc(2) = MAX(0.001_wp,EPSILON(0.0_wp)**0.2_wp) ! the default

      DO i = 1, 2

        CALL nag_quad_monte_carlo(f,a,b,result,rel_acc=rel_acc(i), &
         num_fun_eval=num_fun_eval,repeat_rand=.TRUE.)

        WRITE (nag_std_out,'(/,1X,A,F9.5)') &
         'requested relative accuracy  . . . . . =', rel_acc(i)
        WRITE (nag_std_out,'(1X,A,I9)') &
         'number of function calls . . . . . . . =', num_fun_eval
        WRITE (nag_std_out,'(1X,A,1F9.5/)') &
         'result - approximation to the integral =', result
      END DO

    END PROGRAM nag_quad_md_ex05
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_quad_md_ex05

 requested relative accuracy  . . . . . =  0.01000
 number of function calls . . . . . . . =    1728
 result - approximation to the integral =  0.57583


 requested relative accuracy  . . . . . =  0.00100
 number of function calls . . . . . . . =   19248
 result - approximation to the integral =  0.57536
```

Mathematical Background

1 Dimensionality

A distinction must be made between cases of moderately low dimensionality (say, up to 4 or 5 dimensions), and those of higher dimensionality. Where the number of dimensions is limited, a one-dimensional method may be applied to each dimension, according to some suitable strategy, and high accuracy may be obtainable (using product rules). However, the number of integrand evaluations rises very rapidly with the number of dimensions, so the accuracy obtainable with an acceptable amount of computational labour is limited; for example a product of 3-point rules in 20 dimensions would require more than 10^9 integrand evaluations. Special techniques such as the Monte Carlo, number theoretic and Sag–Szekeres methods should be used to deal with high dimensions (see Davis and Rabinowitz [1]).

2 Automatic Adaptive Procedures

An automatic adaptive strategy in several dimensions normally involves division of the region into subregions, concentrating the divisions in those parts of the region where the integrand is worst behaved. It is difficult to arrange with any generality for variable limits in the inner integral(s). For this reason, some methods use a region where all the limits are constants; this is called a hyper-rectangle. Integrals over regions defined by variable or infinite limits may be handled by transformation to a hyper-rectangle (see Davis and Rabinowitz [1]).

The method used locally in each subregion produced by the adaptive subdivision process is usually one of three types: Monte Carlo, number theoretic or deterministic. Deterministic methods are usually the most rapidly convergent but are often expensive to use for high dimensionality.

References

[1] Davis P J and Rabinowitz P (1975) *Methods of Numerical Integration* Academic Press

[2] 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

[3] 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

[4] Lautrup B (1971) An adaptive multi-dimensional integration procedure *Proc. 2nd Coll. Advanced Methods in Theoretical Physics, Marseille*

[5] Patterson T N L (1968) On some Gauss and Lobatto based integration formulae *Math. Comput.* **22** 877–881

[6] Patterson T N L (1968) The Optimum Addition of Points to Quadrature Formulae *Math. Comput.* **22** 847–856