NAG Library Function Document

nag_quad_1d_gen_vec_multi_rcomm (d01rac)

Note: *this function uses* **optional parameters** *to define choices in the problem specification and in the details of the algorithm. If you wish to use* default *settings for all of the optional parameters, you need only read Sections 1 to 10 of this document. If, however, you wish to reset some or all of the settings please refer to Section 11 for a detailed description of the specification of the optional parameters.*

1 Purpose

nag_quad_1d_gen_vec_multi_rcomm (d01rac) is a general purpose adaptive integrator which calculates an approximation to a vector of definite integrals $\mathbf{F}$ over a finite range $[a, b]$, given the vector of integrands $\mathbf{f}(x)$.

$$\mathbf{F} = \int_a^b \mathbf{f}(x) dx$$

If the same subdivisions of the range are equally good for functions $f_1(x)$ and $f_2(x)$, because $f_1(x)$ and $f_2(x)$ have common areas of the range where they vary slowly and where they vary quickly, then we say that $f_1(x)$ and $f_2(x)$ are 'similar'. nag_quad_1d_gen_vec_multi_rcomm (d01rac) is particularly effective for the integration of a vector of similar functions.

2 Specification

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

```
void nag_quad_1d_gen_vec_multi_rcomm (Integer *irevcm, Integer ni, double a,
     double b, Integer *sid, Integer needi[], double x[], Integer lenx,
     Integer *nx, const double fm[], Integer ldfm, double dinest[],
     double errest[], const Integer iopts[], const double opts[],
     Integer icom[], Integer licom, double com[], Integer lcom,
     NagError *fail)
```

3 Description

nag_quad_1d_gen_vec_multi_rcomm (d01rac) is an extension to various QUADPACK routines, including QAG, QAGS and QAGP. The extensions made allow multiple integrands to be evaluated simultaneously, using a vectorized interface and reverse communication.

The quadrature scheme employed by nag_quad_1d_gen_vec_multi_rcomm (d01rac) can be chosen by you. Six Gauss–Kronrod schemes are available. The algorithm incorporates a global acceptance criterion (as defined by Malcolm and Simpson (1976)), optionally together with the ϵ-algorithm (see Wynn (1956)) to perform extrapolation. The local error estimation is described in Piessens *et al.* (1983).

nag_quad_1d_gen_vec_multi_rcomm (d01rac) is the integration function in the suite of functions nag_quad_1d_gen_vec_multi_rcomm (d01rac) and nag_quad_1d_gen_vec_multi_dimreq (d01rcc). It also uses optional parameters, which can be set and queried using the functions nag_quad_opt_set (d01zkc) and nag_quad_opt_get (d01zlc) respectively. The options available are described in Section 11.

First, the option arrays **iopts** and **opts** must be initialized using nag_quad_opt_set (d01zkc). Thereafter any required options must be set before calling nag_quad_1d_gen_vec_multi_rcomm (d01rac), or the function nag_quad_1d_gen_vec_multi_dimreq (d01rcc).

A typical usage of this suite of functions is (in pseudo-code for clarity),

Setup phase

```
liopts = 100
 lopts = 100
```

```
    allocate(iopts(liopts),opts(lopts))
    d01zkc('initialize = d01rac',iopts,liopts,opts,lopts,fail)
    d01zkc('option = value',iopts,liopts,opts,lopts,fail)
    ...
    d01rcc(ni,lenxrq,ldfmrq,sdfmrq,licmin,licmax,lcmin,lcmax,
           iopts,opts,fail)
    lenx = lenxrq
    ldfm = ldfmrq
    sdfm = sdfmrq
    licom = licmax
    lcom = lcmax
    allocate(icom(licom),com(lcom),x(lenx),fm(ldfm,sdfm),needi(ni),
             dinest(ni),errest(ni))
```

Solve phase

```
   irevcm = 1
    while irevcm≠0
        d01rac(irevcm,ni,a,b,sid,needi,x,lenx,nx,fm,ldfm,
               dinest,errest,iopts,opts,icom,licom,com,
               lcom,fail)
        select case(irevcm)
        case(11)
              Initial solve phase
              evaluate fm(1:ni,1:nx)
        case(12)
              Adaptive solve phase
              evaluate fm(needi(1:ni)=1,1:nx)
        case(0)
              investigate fail
        end select
    end while
```

Diagnostic phase

```
   d01zlc('option',ivalue,rvalue,cvalue,optype,iopts,opts,fail)
    ...
```

During the initial solve phase, the first estimation of the definite integral and error estimate is constructed over the interval $[a,b]$. This will have been divided into s_{pri} level 1 segments, where s_{pri} is the number of **Primary Divisions**, and will use any provided break-points if **Primary Division Mode** $=$ MANUAL.

Once a complete integral estimate over $[a,b]$ is available, i.e., after all the estimates for the level 1 segments have been evaluated, the function enters the adaptive phase. The estimated errors are tested against the requested tolerances ϵ_a and ϵ_r, corresponding to the **Absolute Tolerance** and **Relative Tolerance** respectively. Should this test fail, and additional subdivision be allowed, a segment is selected for subdivision, and is subsequently replaced by two new segments at the next level of refinement. How this segment is chosen may be altered by setting **Prioritize Error** to either favour the segment with the maximum error, or the segment with the lowest level supporting an unacceptable (although potentially non-maximal) error. Up to $\max_{sdiv}$ subdivisions are allowed if sufficient memory is provided, where $\max_{sdiv}$ is the value of **Maximum Subdivisions**.

Once a sufficient number of error estimates have been constructed for a particular integral, the function may optionally use **Extrapolation** to attempt to accelerate convergence. This may significantly lower the amount of work required for a given integration. To minimize the risk of premature convergence from extrapolation, a safeguard ϵ_{safe} can be set using **Extrapolation Safeguard**, and the extrapolated solution will only be considered if $\epsilon_{safe}\epsilon_q \leq \epsilon_{ex}$, where ϵ_q and ϵ_{ex} are the estimated error directly from the quadrature and from the extrapolation respectively. If extrapolation is successful for the computation of integral j, the extrapolated solution will be returned in **dinest**$[j-1]$ on completion of nag_quad_1d_gen_vec_multi_rcomm (d01rac). Otherwise the direct solution will be returned in **dinest**$[j-1]$. This is indicated by the value of **needi**$[j-1]$ on completion.

4 References

Malcolm M A and Simpson R B (1976) Local versus global strategies for adaptive quadrature *ACM Trans. Math. Software* **1** 129–146

Piessens R (1973) An algorithm for automatic integration *Angew. Inf.* **15** 399–401

Piessens R, de Doncker–Kapenga E, Ûberhuber C and Kahaner D (1983) *QUADPACK, A Subroutine Package for Automatic Integration* Springer–Verlag

Wynn P (1956) On a device for computing the $e_m(S_n)$ transformation *Math. Tables Aids Comput.* **10** 91–96

5 Arguments

Note: this function uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the argument **irevcm**. Between intermediate exits and re-entries, **all arguments other than irevcm, needi and fm must remain unchanged**.

Where **FM**(j,i) appears in this document it refers to the array element **fm**$[(i-1)\times$ **ldfm** $+ j - 1]$.

1: **irevcm** – Integer * *Input/Output*

On initial entry: **irevcm** $= 1$.

irevcm $= 1$

Sets up data structures in **icom** and **com** and starts a new integration.

Constraint: **irevcm** $= 1$ on initial entry.

On intermediate exit: **irevcm** $= 11$ or 12.

irevcm requests the integrands $f_j(x_i)$ be evaluated for all required $j \in 1,\dots,n_i$ as indicated by **needi**, and at all the points x_i, for $i = 1,2,\dots,n_x$. Abscissae x_i are provided in **x**$[i-1]$ and $f_j(x_i)$ must be returned in **FM**(j,i).

During the initial solve phase:

irevcm $= 11$

Function values are required to construct the initial estimates of the definite integrals.

If **needi**$[j-1] = 1$, $f_j(x_i)$ must be supplied in **FM**(j,i). This will be the case unless you have abandoned the evaluation of specific integrals on a previous call.

If **needi**$[j-1] = 0$, you have previously abandoned the evaluation of integral j, and hence should not supply the value of $f_j(x_i)$.

dinest and **errest** contain incomplete information during this phase. As such you should not abandon the evaluation of any integrals during this phase unless you do not require their estimate.

If **irevcm** is set to a negative value during this phase, **needi**$[j-1]$, for $j = 1,2,\dots,n_i$, will be set to this negative value and **fail.code** = NE_USER_STOP will be returned.

During the adaptive solve phase:

irevcm $= 12$

Function values are required to improve the estimates of the definite integrals.

If **needi**$[j-1] = 0$, any evaluation of $f_j(x_i)$ will be discarded, so there is no need to provide them.

If **needi**$[j-1] = 1$, $f_j(x_i)$ must be provided in **FM**(j,i).

If **needi**$[j-1] = 2$, 3 or 4, the current error estimate of integral j does not require integrand j to be evaluated and provided in **FM**(j,i). Should you choose to, integrand j can be evaluated in which case **needi**$[j-1]$ must be set to 1.

dinest and **errest** contain complete information during this phase.

If **irevcm** is set to a negative value during this phase **fail.code** = NE_ACCURACY or NE_QUAD_BAD_SUBDIV_INT will be returned and the elements of **needi** will reflect the current state of the adaptive process.

On intermediate re-entry: **irevcm** should normally be left unchanged. However, if **irevcm** is set to a negative value, nag_quad_1d_gen_vec_multi_rcomm (d01rac) will terminate, (see **irevcm** $= 11$ and **irevcm** $= 12$ above).

On final exit: **irevcm** $= 0$.

irevcm $= 0$

Indicates the algorithm has completed.

2: **ni** – Integer *Input*

On entry: n_i, the number of integrands.

3: **a** – double *Input*

On entry: a, the lower bound of the domain.

4: **b** – double *Input*

On entry: b, the upper bound of the domain.

If $|b - a| < 10\epsilon$, where ϵ is the ***machine precision*** (see nag_machine_precision (X02AJC)), then nag_quad_1d_gen_vec_multi_rcomm (d01rac) will return **dinest**$[j-1]$ = **errest**$[j-1] = 0.0$, for $j = 1, 2, \ldots, n_i$.

5: **sid** – Integer * *Output*

For advanced users.

On intermediate exit: **sid** identifies a specific set of abscissae, **x**, returned during the integration process. When a new set of abscissae are generated the value of **sid** is incremented by 1. Advanced users may store calculations required for an identified set **x**, and reuse them should nag_quad_1d_gen_vec_multi_rcomm (d01rac) return the same value of **sid**, i.e., the same set of abscissae was used.

6: **needi**[**ni**] – Integer *Input/Output*

On initial entry: need not be set.

On intermediate exit: **needi**$[j-1]$ indicates what action must be taken for integral $j = 1, 2, \ldots n_i$ (see **irevcm**).

needi$[j-1] = 0$

Do not provide $f_j(x_i)$. Any provided values will be ignored.

needi$[j-1] = 1$

The values $f_j(x_i)$ must be provided in **FM**(j, i), for $i = 1, 2, \ldots, n_x$.

needi$[j-1] = 2$

The values $f_j(x_i)$ are not required, however the error estimate for integral j is still above the requested tolerance. If you wish to provide values for the evaluation of integral j, set **needi**$[j-1] = 1$, and supply $f_j(x_i)$ in **FM**(j, i), for $i = 1, 2, \ldots, n_x$.

needi$[j-1] = 3$

The error estimate for integral j cannot be improved to below the requested tolerance directly, either because no more new splits may be performed due to exhaustion, or due to the detection of extremely bad integrand behaviour. However, providing the values $f_j(x_i)$ may still lead to some improvement, and may lead to an acceptable error estimate indirectly using Wynn's epsilon algorithm. If you wish to provide values for the evaluation of integral j, set **needi**$[j-1] = 1$, and supply $f_j(x_i)$ in **FM**(j, i), for $i = 1, 2, \ldots, n_x$.

needi$[j-1]=4$
: The error estimate of integral j is below the requested tolerance. If you believe this to be false, if for example the result in **dinest**$[j-1]$ is greatly different to what you may expect, you may force the algorithm to re-evaluate this conclusion by including the evaluations of integrand j at x_i, for $i=1,2,\ldots,n_x$, and setting **needi**$[j-1]=1$. Integral and error estimation will be performed again during the next iteration.

On intermediate re-entry: **needi**$[j-1]$ may be used to indicate what action you have taken for integral j.

needi$[j-1]=1$
: You have provided the values $f_j(x_i)$ in $\mathbf{FM}(j,i)$, for $i=1,2,\ldots,n_x$.

needi$[j-1]<0$
: You are abandoning the evaluation of integral j. The current values of **dinest**$[j-1]$ and **errest**$[j-1]$ will be returned on final completion.

Otherwise you have not provided the value $f_j(x_i)$.

On final exit: **needi**$[j-1]$ indicates the final state of integral j.

needi$[j-1]=0$
: The error estimate for F_j is below the requested tolerance.

needi$[j-1]=1$
: The error estimate for F_j is below the requested tolerance after extrapolation.

needi$[j-1]=2$
: The error estimate for F_j is above the requested tolerance.

needi$[j-1]=3$
: The error estimate for F_j is above the requested tolerance, and extremely bad behaviour of integral j has been detected.

needi$[j-1]<0$
: You prohibited further evaluation of integral j.

7: **x**[**lenx**] – double *Input/Output*

On initial entry: if **Primary Division Mode** $=$ AUTOMATIC, **x** need not be set. This is the default behaviour.

If **Primary Division Mode** $=$ MANUAL, **x** is used to supply a set of initial 'break-points' inside the domain of integration. Specifically, **x**$[i-1]$ must contain a break-point x_i^0, for $i=1,2,\ldots,(s_{pri}-1)$, where s_{pri} is the number of **Primary Divisions**.

Constraint: if break-points are supplied, $x_i^0\in(a,b)$, $\left|x_i^0-a\right|>10.0\epsilon$, $\left|x_i^0-b\right|>10.0\epsilon$, for $i=1,2,\ldots,(s_{pri}-1)$.

On intermediate exit: **x**$[i-1]$ is the abscissa x_i, for $i=1,2,\ldots,n_x$, at which the appropriate integrals must be evaluated.

8: **lenx** – Integer *Input*

On entry: the dimension of the array **x**. Currently **lenx** $=\max(122,s_{pri}-1)$ will be sufficient for all cases.

Constraint: **lenx** $\geq$ *lenxrq*, where *lenxrq* is dependent upon the options currently set (see Section 11). *lenxrq* is returned as **lenxrq** from nag_quad_1d_gen_vec_multi_dimreq (d01rcc).

9: **nx** – Integer * *Input/Output*

On initial entry: need not be set.

On intermediate exit: n_x, the number of abscissae at which integrands are required.

On intermediate re-entry: must not be changed.

10: **fm**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **fm** must be at least **ldfm** $\times$ *sdfmrq*, where *sdfmrq* is dependent upon n_i and the options currently set. *sdfmrq* is returned as **sdfmrq** from nag_quad_1d_gen_vec_multi_dimreq (d01rcc). If default options are chosen, *sdfmrq* = *lenxrq*.

On initial entry: need not be set.

On intermediate re-entry: if indicated by **needi**[$j-1$] you must supply the values $f_j(x_i)$ in **FM**(j, i), for $i = 1, 2, \ldots, n_x$ and $j = 1, 2, \ldots, n_i$.

11: **ldfm** – Integer *Input*

On entry: the stride separating matrix row elements in the array **fm**.

Constraint: **ldfm** $\geq$ *ldfmrq*, where *ldfmrq* is dependent upon n_i and the options currently set. *ldfmrq* is returned as **ldfmrq** from nag_quad_1d_gen_vec_multi_dimreq (d01rcc). If default options are chosen, *ldfmrq* = n_i, implying **ldfm** $\geq$ **ni**.

12: **dinest**[**ni**] – double *Input/Output*

dinest[$j-1$] contains the current estimate of the definite integral F_j.

On initial entry: need not be set.

On intermediate re-entry: must not be altered.

On exit: contains the current estimates of the **ni** integrals. If **irevcm** = 0, this will be the final solution.

13: **errest**[**ni**] – double *Input/Output*

errest[$j-1$] contains the current error estimate of the definite integral F_j.

On initial entry: need not be set.

On intermediate re-entry: must not be altered.

On exit: contains the current error estimates for the **ni** integrals. If **irevcm** = 0, **errest** contains the final error estimates of the **ni** integrals.

14: **iopts**[**100**] – const Integer *Communication Array*
15: **opts**[**100**] – const double *Communication Array*

The arrays **iopts** and **opts** MUST NOT be altered between calls to any of the functions nag_quad_1d_gen_vec_multi_rcomm (d01rac), nag_quad_1d_gen_vec_multi_dimreq (d01rcc), nag_quad_opt_set (d01zkc) and nag_quad_opt_get (d01zlc).

16: **icom**[**licom**] – Integer *Communication Array*

icom contains details of the integration procedure, including information on the integration of the n_i integrals over individual segments. This data is stored sequentially in the order that segments are created. For further information see Section 9.1.

17: **licom** – Integer *Input*

On entry: the dimension of the array **icom**.

Constraint: **licom** $\geq$ *licmin*, where *licmin* is dependent upon **ni** and the current options set. *licmin* is returned as **licmin** from nag_quad_1d_gen_vec_multi_dimreq (d01rcc). If the default options are set, then *licmin* = $55 + 6 \times$ **ni**. Larger values than *licmin* are recommended if you anticipate that any integrals will require the domain to be further subdivided.

The maximum value that may be required, *licmax*, is returned as **licmax** from nag_quad_1d_gen_vec_multi_dimreq (d01rcc). If default options are chosen, except for possibly increasing the value of s_{pri}, then *licmax* = $50 + 5 \times$ **ni** + $\left(s_{pri} + 100\right) \times (5 +$ **ni**$)$.

18: **com**[**lcom**] – double *Communication Array*

com contains details of the integration procedure, including information on the integration of the n_i integrals over individual segments. This data is stored sequentially in the order that segments are created. For further information see Section 9.1.

19: **lcom** – Integer *Input*

On entry: the dimension of the array **com**.

Constraint: **lcom** $> lcmin$, where $lcmin$ is dependent upon **ni**, s_{pri} and the current options set. $lcmin$ is returned as **lcmin** from nag_quad_1d_gen_vec_multi_dimreq (d01rcc). If default options are set, then $lcmin = 96 + 12 \times$ **ni**. Larger values are recommended if you anticipate that any integrals will require the domain to be further subdivided.

Given the current options and arguments, the maximum value, $lcmax$, of **lcom** that may be required, is returned as **lcmax** from nag_quad_1d_gen_vec_multi_dimreq (d01rcc). If default options are chosen, $lcmax = 94 + 9 \times \mathbf{ni} + \lceil \mathbf{ni}/2 \rceil + \left(s_{pri} + 100\right) \times (2 + \lceil \mathbf{ni}/2 \rceil + 2 \times \mathbf{ni})$.

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

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ACCURACY

At least one error estimate exceeded the requested tolerances.

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

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

NE_INT

irevcm had an illegal value.
On entry, **irevcm** $= \langle value\rangle$.

On entry, **ni** $= \langle value\rangle$.
Constraint: **ni** ≥ 1.

NE_INT_2

ldfm $< ldfmrq$. If default options are chosen, this implies **ldfm** $<$ **ni**.
On entry, **ldfm** $= \langle value\rangle$.
Constraint: **ldfm** $\geq \langle value\rangle$.

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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_INVALID_ARRAY

On entry, one of **icom** and **com** has become corrupted.

NE_INVALID_OPTION

Either the option arrays **iopts** and **opts** have not been initialized for nag_quad_1d_gen_vec_multi_rcomm (d01rac), or they have become corrupted.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_QUAD_BAD_SUBDIV_INT

Extremely bad behaviour was detected for at least one integral.

Extremely bad behaviour was detected for at least one integral. At least one other integral error estimate was above the requested tolerance.

NE_QUAD_BRKPTS_INVAL

On entry, **Primary Division Mode** $=$ MANUAL and at least one supplied break-point in **x** is outside of the domain of integration.

NE_TOO_SMALL

lcom is insufficient for additional subdivision.
On entry, **lcom** $= \langle value \rangle$.
Constraint: **lcom** $\geq \langle value \rangle$.

lenx is insufficient for the chosen options.
On entry, **lenx** $= \langle value \rangle$.
Constraint: **lenx** $\geq \langle value \rangle$.

licom is insufficient for additional subdivision.
On entry, **licom** $= \langle value \rangle$.
Constraint: **licom** $\geq \langle value \rangle$.

NE_USER_STOP

Evaluation of all integrals has been stopped during the initial phase.

7 Accuracy

nag_quad_1d_gen_vec_multi_rcomm (d01rac) cannot guarantee, but in practice usually achieves, the following accuracy for each integral F_j:

$$\left|F_j - \mathbf{dinest}[j-1]\right| \leq \mathrm{tol}$$

where

$$\mathrm{tol} = \max\left(\epsilon_a, \epsilon_r \times \left|F_j\right|\right)$$

ϵ_a and ϵ_r are the error tolerances **Absolute Tolerance** and **Relative Tolerance** respectively. Moreover, it returns **errest**, the entries of which in normal circumstances satisfy,

$$\left|F_j - \mathbf{dinest}[j-1]\right| \leq \mathbf{errest}[j-1] \leq \mathrm{tol}.$$

8 Parallelism and Performance

nag_quad_1d_gen_vec_multi_rcomm (d01rac) is threaded by NAG for parallel execution in multi-threaded implementations of the NAG Library.

nag_quad_1d_gen_vec_multi_rcomm (d01rac) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The time required by nag_quad_1d_gen_vec_multi_rcomm (d01rac) is usually dominated by the time required to evaluate the values of the integrands f_j.

nag_quad_1d_gen_vec_multi_rcomm (d01rac) will be most efficient if any badly behaved integrands provided have irregularities over similar subsections of the domain. For example, evaluation of the integrals,

$$\int_0^1 \begin{pmatrix} \log(x) \\ x^{-\frac{1}{2}} \\ x^2 \end{pmatrix} dx$$

will be quite efficient, as the irregular behaviour of the first two integrands is at $x = 0$. On the contrary, the evaluation of the integrals,

$$\int_0^1 \begin{pmatrix} \log(x) \\ \log(1-x) \end{pmatrix} dx$$

will be less efficient, as the two integrands have singularities at opposite ends of the domain, which will result in subdivisions which are only of use to one integrand. In such cases, it will be more efficient to use two sets of calls to nag_quad_1d_gen_vec_multi_rcomm (d01rac).

nag_quad_1d_gen_vec_multi_rcomm (d01rac) will flag extremely bad behaviour if a sub-interval $\bar{k}$ with bounds $[a_{\bar{k}}, b_{\bar{k}}]$ satisfying $|b_{\bar{k}} - a_{\bar{k}}| < \max(\delta_a, \delta_r \times |b - a|)$ has a local error estimate greater than the requested tolerance for at least one integral. The values δ_a and δ_r can be set through the optional parameters **Absolute Interval Minimum** and **Relative Interval Minimum** respectively.

9.1 Details of the Computation

This section is recommended for expert users only. It describes the contents of the arrays **com** and **icom** upon exit from nag_quad_1d_gen_vec_multi_rcomm (d01rac) with **fail.code** = NE_NOERROR, NE_ACCURACY or NE_QUAD_BAD_SUBDIV_INT, and provided at least one iteration completed, failure due to insufficient **licom** or **lcom**.

The arrays **icom** and **com** contain details of the integration, including various scalars, one-dimensional arrays, and (effectively) two-dimensional arrays. The dimensions of these arrays vary depending on the arguments and options used and the progress of the algorithm. Here we describe some of these details, including how and where they are stored in **icom** and **com**.

Scalar quantities:

The indices in **icom** including the following scalars are available via query only options, see Section 11.2. For example, I_{ldi} is the integer value returned by the option **Index LDI**. Note the indices returned start at 1 and so the corresponding zero based indices require 1 to be subtracted as shown below. This is also true for the location of arrays within **icom** and **com** and consequently the indices of the elements of the one- and two-dimensional arrays must be modified as detailed below.

ldi The leading dimension of the two-dimensional integer arrays stored in **icom** detailed below.
$ldi =$ **icom**$[I_{ldi} - 1]$.

ldr The leading dimension of the two-dimensional real arrays stored in **com** detailed below.
$ldr =$ **icom**$[I_{ldr} - 1]$.

nsdiv The number of segments that have been subdivided during the adaptive process.
$nsdiv =$ **icom**$[I_{nsdiv} - 1]$.

nseg The total number of segments formed.
$nseg = 2nsdiv + s_{pri}$.
$nseg =$ **icom**$[I_{nseg} - 1]$.

dsp The reference of the first element of the array *ds* stored in **com**.
$dsp = \mathbf{icom}[I_{dsp} - 1]$.

esp The reference of the first element of the array *es* stored in **com**.
$esp = \mathbf{icom}[I_{esp} - 1]$.

evalsp The reference of the first element of the array *evals* stored in **icom**.
$evalsp = \mathbf{icom}[I_{evalsp} - 1]$.

fcp The reference of the first element of the array *fcount* stored in **icom**.
$fcp = \mathbf{icom}[I_{fcp} - 1]$.

sinforp The reference of the first element of the array *sinfor* stored in **com**.
$sinforp = \mathbf{icom}[I_{sinforp} - 1]$.

sinfoip The reference of the first element of the array *sinfoi* stored in **icom**.
$sinfoip = \mathbf{icom}[I_{sinfoip} - 1]$.

One-dimensional arrays:

$fcount[n_i]$
$fcount[0] = \mathbf{icom}[fcp - 1]$.

$fcount[j - 1]$ contains the number of different approximations of integral j calculated, for $j = 1, 2, \dots, n_i$.

Two-dimensional arrays:

$sinfoi[5 \times nseg]$
$sinfoi[0] = \mathbf{icom}[sinfoip - 1]$.
sinfoi contains information about the hierarchy of splitting.

$sinfoi[(k - 1) \times ldi]$ contains the split identifier for segment k, for $k = 1, 2, \dots, nseg$.

$sinfoi[(k - 1) \times ldi + 1]$ contains the parent segment number of segment k (i.e., the segment was split to create segment k), for $k = 1, 2, \dots, nseg$.

$sinfoi[(k - 1) \times ldi + 2]$ and $sinfoi[(k - 1) \times ldi + 3]$ contain the segment numbers of the two child segments formed from segment k, if segment k has been split. If segment k has not been split, these will be negative.

$sinfoi[(k - 1) \times ldi + 4]$ contains the level at which the segment exists, corresponding to $n_a + 1$, where n_a is the number of ancestor segments of segment k, for $k = 1, 2, \dots, nseg$. A negative level indicates that segment k will not be split further, the level is then given by the absolute value of $sinfoi[(k - 1) \times ldi + 4]$.

$sinfor[2 \times nseg]$
$sinfor[0] = \mathbf{com}[sinforp - 1]$.
sinfor contains the bounds of each segment.

$sinfor[(k - 1) \times ldr]$ contains the lower bound of segment k, for $k = 1, 2, \dots, nseg$.

$sinfor[(k - 1) \times ldr + 1]$ contains the upper bound of segment k, for $k = 1, 2, \dots, nseg$.

$evals[n_i \times nseg]$
$evals[0] = \mathbf{icom}[evalsp - 1]$.

evals contains information to indicate whether an estimate of the integral j has been obtained over segment k, and if so whether this evaluation still contributes to the direct estimate of F_j, for $j = 1, 2, \dots, n_i$ and $k = 1, 2, \dots, nseg$.

$evals[(k - 1) \times ldi + j - 1] = 0$ indicates that integral j has not been evaluated over segment k.

$evals[(k - 1) \times ldi + j - 1] = 1$ indicates that integral j has been evaluated over segment k, and that this evaluation contributes to the direct estimate of F_j.

$evals[(k - 1) \times ldi + j - 1] = 2$ indicates that integral j has been evaluated over segment k, that this evaluation contributes to the direct estimate of F_j, and that you have requested no further evaluation of this integral at this segment by setting $\mathbf{needi}[j - 1] < 0$.

$evals[(k-1)\times ldi+j-1]=3$ indicates that integral j has been evaluated over segment k, and this evaluation no longer contributes to the direct estimate of F_j.

$evals[(k-1)\times ldi+j-1]=4$ indicates that integral j has been evaluated over segment k, that this evaluation contributes to the direct estimate of F_j, and that this segment is too small for any further splitting to be performed. Integral j also has a local error estimate over this segment above the requested tolerance. Such segments cause nag_quad_1d_gen_vec_multi_rcomm (d01rac) to return **fail.code** = NE_QUAD_BAD_SUBDIV_INT, indicating extremely bad behaviour.

$evals[(k-1)\times ldi+j-1]=5$ indicates that integral j has been evaluated over segment k, that this evaluation contributes to the direct estimate of F_j, and that this segment is too small for any further splitting to be performed. The local error estimate is however below the requested tolerance.

$ds[n_i\times nseg]$

$ds[0]=\mathbf{com}[dsp-1]$.

$ds[(k-1)\times ldr+j-1]$ contains the definite integral estimate of the jth integral over the kth segment, $ds_{j,k}$, provided it has been evaluated, for $j=1,2,\ldots,n_i$ and $k=1,2,\ldots,nseg$.

$es[n_i\times nseg]$

$es[0]=\mathbf{com}[esp-1]$.

$es[(k-1)\times ldr+j-1]$ contains the definite integral error estimate of the jth integral over the kth segment, $es_{j,k}$, provided it has been evaluated, for $j=1,2,\ldots,n_i$ and $k=1,2,\ldots,nseg$.

For each integral j, the direct approximation D_j of F_j, and its error estimate E_j, may be constructed as,

$$F_j\approx D_j=\sum_{K_j} ds_{j,k},$$
$$\left|F_j-D_j\right|\approx E_j=\sum_{K_j} es_{j,k},$$

where K_j is the set of all contributing segments, $K_j=\{k\mid evals[(k-1)\times ldi+j-1]=1,\,2,\,4\text{ or }5,\ 1\le k\le nseg\}$. D_j will have been returned in **dinest**$[j-1]$, unless extrapolation was successful, as indicated by **needi**$[j-1]$.

Similarly, E_j will have been returned in **errest**$[j-1]$ unless extrapolation was successful, in which case the error estimate from the extrapolation will have been returned. If for a given integral j one or more contributing segments have unacceptable error estimates, it may be possible to improve the direct approximation by replacing the contributions from these segments with more accurate estimates should these be calculable by some means. Indeed for any segment $\bar{k}\in k$, with lower bound $a_{\bar{k}}=sinfor\left(1,\bar{k}\right)$ and upper bound $b_{\bar{k}}=sinfor\left(2,\bar{k}\right)$, one may alter the direct approximation D_j by the following,

$$ds_{j,\bar{k}}^{\text{new}}\approx\int_{a_{\bar{k}}}^{b_{\bar{k}}} f_j(x)\ dx$$
$$D_j=\sum_{K_j} ds_{j,k}-ds_{j,\bar{k}}+ds_{j,\bar{k}}^{\text{new}}.$$

The error estimate E_j may be altered similarly.

10 Example

This example integrates

$$\mathbf{F}=\int_0^{\pi}\begin{pmatrix} x\sin(2x)\cos(15x)\\ x^2\sin(2x)\cos(50x)\end{pmatrix}\quad dx.$$

10.1 Program Text

```
/* nag_quad_1d_gen_vec_multi_rcomm (d01rac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd01.h>
#include <nagx01.h>

/* Print information on splitting and evaluations over subregions? */
Nag_Boolean disp_integration_info = Nag_TRUE;

static void display_integration_details(const Integer ni,
                                        const Integer iopts[],
                                        const double opts[],
                                        const Integer icom[],
                                        const double com[]);
static void display_option(const char *optstr, const Nag_VariableType optype,
                           const Integer ivalue, const double rvalue,
                           const char *cvalue);

int main(void)
{
#define FM(J,I) fm[(I-1)*ldfm + J-1]

  /* Scalars */
  int exit_status = 0;
  Integer len_cvalue;
  double a, b, rvalue;
  Integer irevcm, ivalue, i, j, lcmax, lcmin, lcom, ldfm, ldfmrq,
         lenx, lenxrq, licmax, licmin, licom, liopts, lopts, ni, nx,
         sdfm, sdfmrq, sid;
  /* Arrays */
  char cvalue[17];
  double *com = 0, *dinest = 0, *errest = 0, *fm = 0, *opts = 0, *x = 0;
  Integer *icom = 0, *iopts = 0, *needi = 0;

  /* NAG types */
  Nag_VariableType optype;
  NagError fail;

  printf("nag_quad_1d_gen_vec_multi_rcomm (d01rac) Example Program Results"
         "\n\n");

  /* Setup phase. */
  /* Set problem parameters. */
  ni = 2;
  a = 0.0;
  b = nag_pi;

  liopts = 100;
  lopts = 100;
  if (!(opts = NAG_ALLOC((lopts), double)) ||
      !(iopts = NAG_ALLOC((liopts), Integer))
         )
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  INIT_FAIL(fail);
  /* Initialize option arrays using nag_quad_opt_set (d01zkc). */
  nag_quad_opt_set("Initialize = nag_quad_1d_gen_vec_multi_rcomm",
                   iopts, liopts, opts, lopts, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_quad_opt_set (d01zkc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  nag_quad_opt_set("Quadrature Rule = gk41", iopts, liopts, opts, lopts,
                   &fail);
  nag_quad_opt_set("Absolute Tolerance = 1.0e-7", iopts, liopts, opts, lopts,
                   &fail);
  nag_quad_opt_set("Relative Tolerance = 1.0e-7", iopts, liopts, opts, lopts,
                   &fail);

  /* Determine required array dimensions for
   * nag_quad_1d_gen_vec_multi_rcomm (d01rac) using
   * nag_quad_1d_gen_vec_multi_dimreq (d01rcc).
   */
  nag_quad_1d_gen_vec_multi_dimreq(ni, &lenxrq, &ldfmrq, &sdfmrq,
                                   &licmin, &licmax, &lcmin, &lcmax,
                                   iopts, opts, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_quad_1d_gen_vec_multi_dimreq (d01rcc).\n%s\n",
           fail.message);
    exit_status = 2;
    goto END;
  }
  ldfm = ldfmrq;
  sdfm = sdfmrq;
  lenx = lenxrq;
  licom = licmax;
  lcom = lcmax;

  /* Allocate remaining arrays. */
  if (!(x = NAG_ALLOC((lenx), double)) ||
      !(needi = NAG_ALLOC((ni), Integer)) ||
      !(fm = NAG_ALLOC((ldfm) * (sdfm), double)) ||
      !(dinest = NAG_ALLOC((ni), double)) ||
      !(errest = NAG_ALLOC((ni), double)) ||
      !(com = NAG_ALLOC((lcom), double)) ||
      !(icom = NAG_ALLOC((licom), Integer))
        )
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Solve phase. */
  /* Use nag_quad_1d_gen_vec_multi_rcomm (d01rac) to evaluate the
   * definite integrals of:
   *  f_1 = (x*sin(2*x))*cos(15*x)
   *  f_2 = (x*sin(2*x))*(x*cos(50*x))
   */

  INIT_FAIL(fail);
  /* Set initial irevcm. */
  irevcm = 1;
  while (irevcm) {
    /* nag_quad_1d_gen_vec_multi_rcomm (d01rac).
     * One-dimensional quadrature, adaptive, vectorized, multi-integral,
     * reverse communication.
     */
    nag_quad_1d_gen_vec_multi_rcomm(&irevcm, ni, a, b,
                                    &sid, needi, x, lenx, &nx, fm, ldfm,
                                    dinest, errest,
                                    iopts, opts, icom, licom, com, lcom,
                                    &fail);
    switch (irevcm) {
    case 11:
      /* Initial returns.
       * These will occur during the non-adaptive phase.
       * All values must be supplied.
       * dinest and errest do not contain approximations over the complete
       * interval at this stage.
       */

      for (i = 1; i <= nx; i++) {
        FM(2, i) = x[i - 1] * sin(2.0 * x[i - 1]);
        FM(1, i) = FM(2, i) * cos(15.0 * x[i - 1]);
        /* Complete f_2 calculation. */
        FM(2, i) = FM(2, i) * x[i - 1] * cos(50.0 * x[i - 1]);
      }

      break;
    case 12:
      /* Intermediate returns.
       * These will occur during the adaptive phase.
       * All requested values must be supplied.
       * dinest and errest contain approximations over the complete
       * interval at this stage.
       */
      if ((needi[0] == 1) && (needi[1] == 1)) {
        for (i = 1; i <= nx; i++) {
          FM(2, i) = x[i - 1] * sin(2.0 * x[i - 1]);
          FM(1, i) = FM(2, i) * cos(15.0 * x[i - 1]);
          /* Complete f_2 calculation. */
          FM(2, i) = FM(2, i) * x[i - 1] * cos(50.0 * x[i - 1]);
        }
      }
      else if (needi[0] == 1) {
        /* Only calculation of f_1 is requried. */
        for (i = 1; i <= nx; i++)
          FM(1, i) =
                 (x[i - 1] * sin(2.0 * x[i - 1])) * (cos(15.0 * x[i - 1]));
      }
      else if (needi[1] == 1) {
        /* Only calculation of f_2 is requried. */
        for (i = 1; i <= nx; i++)
          FM(2, i) =
                 (x[i - 1] * sin(2.0 * x[i - 1])) * (x[i - 1] *
                                                     cos(50.0 * x[i - 1]));
      }
      break;
    case 0:
      /* Final return. Test fail.code. */
      switch (fail.code) {
      case NE_NOERROR:
        break;
      case NE_ACCURACY:
        printf("Warning: nag_quad_1d_gen_vec_multi_rcomm (d01rac) has "
               "returned at \n least one error estimate exceeding the"
               " requested tolerances\n");
        break;
      case NE_QUAD_BAD_SUBDIV_INT:
        /* Useful information has been returned. */
        printf("Warning: nag_quad_1d_gen_vec_multi_rcomm (d01rac) has "
               "detected \n extremely bad behaviour for at least"
               " one integral\n");
        break;
      default:;
        /* An unrecoverable error has been detected. */
        printf("Error from nag_quad_1d_gen_vec_multi_rcomm (d01rac).\n%s\n",
               fail.message);
        exit_status = 3;
        goto END;
      }
    }
  }

  /* Query some currently set options using nag_quad_opt_get (d01zlc). */
  len_cvalue = 17;
  nag_quad_opt_get("Quadrature rule", &ivalue, &rvalue, cvalue, len_cvalue,
                   &optype, iopts, opts, &fail);
  display_option("Quadrature rule", optype, ivalue, rvalue, cvalue);
  nag_quad_opt_get("Maximum Subdivisions", &ivalue, &rvalue, cvalue,
                   len_cvalue, &optype, iopts, opts, &fail);
  display_option("Maximum Subdivisions", optype, ivalue, rvalue, cvalue);
  nag_quad_opt_get("Extrapolation", &ivalue, &rvalue, cvalue, len_cvalue,
                   &optype, iopts, opts, &fail);
  display_option("Extrapolation", optype, ivalue, rvalue, cvalue);
  nag_quad_opt_get("Extrapolation Safeguard", &ivalue, &rvalue,
                   cvalue, len_cvalue, &optype, iopts, opts, &fail);
  display_option("Extrapolation safeguard", optype, ivalue, rvalue, cvalue);

  /* Print solution. */
  printf("\n Integral |  needi  |   dinest   |   errest   \n");
  for (j = 1; j <= ni; j++)
    printf(" %9" NAG_IFMT " %9" NAG_IFMT " %12.4e %12.4e\n",
           j, needi[j - 1], dinest[j - 1], errest[j - 1]);

  /* Investigate integration strategy. */
  if (disp_integration_info)
    display_integration_details(ni, iopts, opts, icom, com);

END:
  NAG_FREE(com);
  NAG_FREE(dinest);
  NAG_FREE(errest);
  NAG_FREE(fm);
  NAG_FREE(opts);
  NAG_FREE(x);
  NAG_FREE(icom);
  NAG_FREE(iopts);
  NAG_FREE(needi);
  return exit_status;
}

static void display_integration_details(const Integer ni,
                                        const Integer iopts[],
                                        const double opts[],
                                        const Integer icom[],
                                        const double com[])
{
#define FCOUNT(J) icom[fcp + J-2]
#define EVALS(J,K) icom[evalsp +(K-1)*ldi + J-2]
#define SINFOI(L,K) icom[sinfoip +(K-1)*ldi + L-2]
#define DS(J,K) com[dsp + (K-1)*ldr + J-2]
#define ES(J,K) com[esp + (K-1)*ldr + J-2]
#define SINFOR(L,K) com[sinforp + (K-1)*ldr + L-2]
  double lbnd, ubnd, rvalue;
  Integer ldi, ldr, sinfoip, sinforp, evalsp, fcp, dsp, esp, nseg, nsdiv;
  Integer child1, child2, j, k, level, parent, sid, index, len_cvalue;
  char cvalue[17];
  NagError fail;
  Nag_VariableType optype;

  /* Request communication array indices */
  INIT_FAIL(fail);
  len_cvalue = 17;
  nag_quad_opt_get("Index nseg", &index, &rvalue, cvalue, len_cvalue,
                   &optype, iopts, opts, &fail);
  nseg = icom[index - 1];
  nag_quad_opt_get("Index nsdiv", &index, &rvalue, cvalue, len_cvalue,
                   &optype, iopts, opts, &fail);
  nsdiv = icom[index - 1];
  nag_quad_opt_get("Index ldi", &index, &rvalue, cvalue, len_cvalue,
                   &optype, iopts, opts, &fail);
  ldi = icom[index - 1];
  nag_quad_opt_get("Index ldr", &index, &rvalue, cvalue, len_cvalue,
                   &optype, iopts, opts, &fail);
  ldr = icom[index - 1];
  nag_quad_opt_get("Index fcp", &index, &rvalue, cvalue, len_cvalue,
                   &optype, iopts, opts, &fail);
  fcp = icom[index - 1];
  nag_quad_opt_get("Index evalsp", &index, &rvalue, cvalue, len_cvalue,
                   &optype, iopts, opts, &fail);
  evalsp = icom[index - 1];
  nag_quad_opt_get("Index sinfoip", &index, &rvalue, cvalue, len_cvalue,
                   &optype, iopts, opts, &fail);
  sinfoip = icom[index - 1];
  nag_quad_opt_get("Index dsp", &index, &rvalue, cvalue, len_cvalue,
                  &optype, iopts, opts, &fail);
  dsp = icom[index - 1];
  nag_quad_opt_get("Index esp", &index, &rvalue, cvalue, len_cvalue,
                  &optype, iopts, opts, &fail);
  esp = icom[index - 1];
  nag_quad_opt_get("Index sinforp", &index, &rvalue, cvalue, len_cvalue,
                  &optype, iopts, opts, &fail);
  sinforp = icom[index - 1];

  printf("\n Information on integration:\n ni = %3" NAG_IFMT
         ", nseg = %3" NAG_IFMT ", nsdiv = %3" NAG_IFMT ".", ni, nseg, nsdiv);
  for (j = 1; j <= ni; j++)
    printf("\n Integral %2" NAG_IFMT " total approximations: %3" NAG_IFMT ".",
           j, FCOUNT(j));
  printf("\n\n Information on subdivision and evaluations over segments.\n");
  for (k = 1; k <= nseg; k++) {
    printf("\n");
    sid = SINFOI(1, k);
    parent = SINFOI(2, k);
    child1 = SINFOI(3, k);
    child2 = SINFOI(4, k);
    level = SINFOI(5, k);
    lbnd = SINFOR(1, k);
    ubnd = SINFOR(2, k);
    printf(" Segment %3" NAG_IFMT ".\n Sid = %3" NAG_IFMT ", Parent = "
           "%3" NAG_IFMT ", Level = %3" NAG_IFMT ".\n", k, sid, parent,
           level);
    if (child1 > 0)
      printf(" Children = (%3" NAG_IFMT ",%3" NAG_IFMT ").\n", child1,
             child2);
    printf(" Bounds (%11.4e,%11.4e).\n", lbnd, ubnd);
    for (j = 1; j <= ni; j++) {
      if (EVALS(j, k) > 0 && EVALS(j, k) < 5) {
        printf(" Integral %2" NAG_IFMT " approximation : %11.4e.\n", j,
               DS(j, k));
        printf(" Integral %2" NAG_IFMT " error estimate: %11.4e.\n", j,
               ES(j, k));
        if (EVALS(j, k) == 3)
          printf(" Integral %2" NAG_IFMT " evaluation has been superseded by "
                 "descendants.\n", j);
      }
    }
  }
  fflush(stdout);
}

static void display_option(const char *optstr, const Nag_VariableType optype,
                           const Integer ivalue, const double rvalue,
                           const char *cvalue)
{
  /* Query optype and print the appropriate option values. */
  switch (optype) {
  case Nag_Integer:
    printf("   %30s :    %13" NAG_IFMT "\n", optstr, ivalue);
    break;
  case Nag_Real:
    printf("   %30s :    %13.4e\n", optstr, rvalue);
    break;
  case Nag_Character:
    printf("   %30s : %16s\n", optstr, cvalue);
    break;
  case Nag_Integer_Additional:
    printf("   %30s :    %3" NAG_IFMT " %16s\n", optstr, ivalue, cvalue);
    break;
  case Nag_Real_Additional:
    printf("   %30s :    %13.4e %16s\n", optstr, rvalue, cvalue);
    break;
  default:;
  }
  fflush(stdout);
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_quad_1d_gen_vec_multi_rcomm (d01rac) Example Program Results

                  Quadrature rule :             GK41
             Maximum Subdivisions :               50
                    Extrapolation :               ON
          Extrapolation safeguard :       1.0000e-12

 Integral |  needi  |   dinest   |   errest
        1        0  -2.8431e-02   1.1234e-14
        2        0   7.9083e-03   2.6600e-09

 Information on integration:
 ni =   2, nseg =   7, nsdiv =   3.
 Integral  1 total approximations:   2.
 Integral  2 total approximations:   4.

 Information on subdivision and evaluations over segments.

 Segment   1.
 Sid =   1, Parent =   0, Level =   1.
 Children = (  2,  3).
 Bounds ( 0.0000e+00, 3.1416e+00).
 Integral  1 approximation : -2.8431e-02.
 Integral  1 error estimate:  8.0372e-04.
 Integral  1 evaluation has been superseded by descendants.
 Integral  2 approximation : -3.6050e-01.
 Integral  2 error estimate:  4.2596e+00.
 Integral  2 evaluation has been superseded by descendants.

 Segment   2.
 Sid =   2, Parent =   1, Level =   2.
 Children = (  6,  7).
 Bounds ( 0.0000e+00, 1.5708e+00).
 Integral  1 approximation : -1.2285e-03.
 Integral  1 error estimate:  2.8161e-15.
 Integral  2 approximation :  1.9771e-03.
 Integral  2 error estimate:  4.0437e-01.
 Integral  2 evaluation has been superseded by descendants.

 Segment   3.
 Sid =   2, Parent =   1, Level =   2.
 Children = (  4,  5).
 Bounds ( 1.5708e+00, 3.1416e+00).
 Integral  1 approximation : -2.7202e-02.
 Integral  1 error estimate:  8.4182e-15.
 Integral  2 approximation :  5.9313e-03.
 Integral  2 error estimate:  3.0259e+00.
 Integral  2 evaluation has been superseded by descendants.

 Segment   4.
 Sid =   3, Parent =   3, Level =   3.
 Bounds ( 1.5708e+00, 2.3562e+00).
 Integral  2 approximation :  1.0922e-01.
 Integral  2 error estimate:  7.9151e-10.

 Segment   5.
 Sid =   3, Parent =   3, Level =   3.
 Bounds ( 2.3562e+00, 3.1416e+00).
 Integral  2 approximation : -1.0329e-01.
 Integral  2 error estimate:  1.6413e-09.

 Segment   6.
 Sid =   4, Parent =   2, Level =   3.
 Bounds ( 0.0000e+00, 7.8540e-01).
```

```
Integral  2 approximation :  1.2343e-02.
Integral  2 error estimate:  5.2456e-11.

Segment   7.
Sid =   4, Parent =   2, Level =   3.
Bounds ( 7.8540e-01, 1.5708e+00).
Integral  2 approximation : -1.0365e-02.
Integral  2 error estimate:  1.7467e-10.
```

11 Optional Parameters

This section can be skipped if you wish to use the default values for all optional parameters, otherwise, the following is a list of the optional parameters available. A full description of each optional parameter is provided in Section 11.1.

Absolute Interval Minimum

Absolute Tolerance

Extrapolation

Extrapolation Safeguard

Maximum Subdivisions

Primary Division Mode

Primary Divisions

Prioritize Error

Quadrature Rule

Relative Interval Minimum

Relative Tolerance

The following optional parameters, see Section 11.2, may be utilized by expert users in conjunction with the information provided in Section 9.1.

Index LDI

Index LDR

Index NSDIV

Index NSEG

Index FCP

Index EVALSP

Index DSP

Index ESP

Index SINFOIP

Index SINFORP

11.1 Description of the Optional Parameters

For each option, we give a summary line, a description of the optional parameter and details of constraints.

The summary line contains:

the keywords, where the minimum abbreviation of each keyword is underlined;

a parameter value, where the letters a, i and r denote options that take character, integer and real values respectively;

the default value.

The following symbols represent various machine constants:

ϵ represents the ***machine precision*** (see nag_machine_precision (X02AJC));

$e_{\max}$ is the maximum exponent argument of the model of floating-point arithmetic, (see nag_real_max_exponent (X02BLC));

$R_{\max}$ represents the largest representable real value (see nag_real_largest_number (X02ALC)).

All options accept the value 'DEFAULT' in order to return single options to their default states.

Keywords and character values are case insensitive, however they must be separated by at least one space.

Unsetable options will return the appropriate value when calling nag_quad_opt_get (d01zlc). They will have no effect if passed to nag_quad_opt_set (d01zkc).

For nag_quad_1d_gen_vec_multi_rcomm (d01rac) the maximum length of the argument **cvalue** used by nag_quad_opt_get (d01zlc) is 15.

Absolute Interval Minimum r Default $= 128.0\epsilon$

$r = \delta_a$, the absolute lower limit for a segment to be considered for subdivision. See also **Relative Interval Minimum** and Section 9.

Constraint: $r \geq 128\epsilon$.

Absolute Tolerance r Default $= 1024\epsilon$

$r = \epsilon_a$, the absolute tolerance required. See also **Relative Tolerance** and Section 3.

Constraint: $r \geq 0.0$.

Extrapolation a Default $=$ ON

Activate or deactivate the use of the ϵ algorithm (Wynn (1956)). **Extrapolation** often reduces the number of iterations required to achieve the desired solution, but it can occasionally lead to premature convergence towards an incorrect answer.

ON
: Use extrapolation.

OFF
: Disable extrapolation.

Extrapolation Safeguard r Default $= 1.0\text{e}{-12}$

$r = \epsilon_{safe}$. If ϵ_q is the estimated error from the quadrature evaluation alone, and ϵ_{ex} is the error estimate determined using extrapolation, then the extrapolated solution will only be accepted if $\epsilon_{safe}\epsilon_q \leq \epsilon_{ex}$.

Maximum Subdivisions i Default $= 50$

$i = \max_{sdiv}$, the maximum number of subdivisions the algorithm may use in the adaptive phase, forming at most an additional $(2 \times \max_{sdiv})$ segments.

Primary Divisions i Default $= 1$

$i = s_{pri}$, the number of initial segments of the domain $[a, b]$. By default the initial segment is the entire domain.

Constraint: $0 < i < 1000000$.

Primary Division Mode a Default $=$ AUTOMATIC

Determines how the initial set of s_{pri} segments will be generated.

AUTOMATIC
: nag_quad_1d_gen_vec_multi_rcomm (d01rac) will automatically generate s_{pri} segments of equal size covering the interval $[a, b]$.

MANUAL

nag_quad_1d_gen_vec_multi_rcomm (d01rac) will use the break-points x_i^0, for $i = 1, 2, \ldots, s_{pri} - 1$, supplied in **x** on initial entry to generate the initial segments covering $[a, b]$. These may be supplied in any order, however it will be more efficient to supply them in ascending (or descending if $a > b$) order. Repeated break-points are allowed, although this will generate fewer initial segments.

Note: an absolute bound on the size of an initial segment of 10.0ϵ is automatically applied in all cases, and will result in fewer initial subdivisions being generated if automatically generated or supplied break-points result in segments smaller than this.

Prioritize Error a Default = LEVEL

Indicates how new subdivisions of segments sustaining unacceptable local errors for integrals should be prioritized.

LEVEL

Segments with lower level with unsatisfactory error estimates will be chosen over segments with greater error on higher levels. This will probably lead to more integrals being improved in earlier iterations of the algorithm, and hence will probably lead to fewer repeated returns (see argument **sid**), and to more integrals being satisfactorily estimated if computational exhaustion occurs.

MAXERR

The segment with the worst overall error will be split, regardless of level. This will more rapidly improve the worst integral estimates, although it will probably result in the fewest integrals being improved in earlier iterations, and may hence lead to more repeated returns (see argument **sid**), and potentially fewer integrals satisfying the requested tolerances if computational exhaustion occurs.

Quadrature Rule a Default = GK15

The basic quadrature rule to be used during the integration. Currently 6 Gauss–Kronrod rules are available, all identifiable by the letters GK followed by the number of points required by the Kronrod rule. Higher order rules generally provide higher accuracy with fewer subdivisons. However, for integrands with sharp singularities, lower order rules may be more efficient, particularly if the integrand away from the singularity is well behaved. With higher order rules, you may need to increase the **Absolute Interval Minimum** and the **Relative Interval Minimum** to maintain numerical difference between the abscissae and the segment bounds.

GK15

The Gauss–Kronrod rule based on 7 Gauss points and 15 Kronrod points.

GK21

The Gauss–Kronrod rule based on 10 Gauss points and 21 Kronrod points. This is the rule used by nag_1d_quad_gen_1 (d01sjc).

GK31

The Gauss–Kronrod rule based on 15 Gauss points and 31 Kronrod points.

GK41

The Gauss–Kronrod rule based on 20 Gauss points and 41 Kronrod points.

GK51

The Gauss–Kronrod rule based on 25 Gauss points and 51 Kronrod points.

GK61

The Gauss–Kronrod rule based on 30 Gauss points and 61 Kronrod points. This is the highest order rule, most suitable for highly oscilliatory integrals.

Relative Interval Minimum r Default = 1.0e−6

$r = \delta_r$, the relative factor in the lower limit, $\delta_r|b - a|$, for a segment to be considered for subdivision. See also **Absolute Interval Minimum** and Section 9.

Constraint: $r \geq 0.0$.

Relative Tolerance r Default $= \sqrt{\epsilon}$

$r = \epsilon_r$, the required relative tolerance. See also **Absolute Tolerance** and Section 3.

Constraint: $r \geq 0.0$.

Note: setting both $\epsilon_r = \epsilon_a = 0.0$ is possible, although it will most likely result in an excessive amount of computational effort.

11.2 Diagnostic Options

These options are provided for expert users who wish to examine and modify the precise details of the computation. They should only be used **after** nag_quad_1d_gen_vec_multi_rcomm (d01rac) returns, as opposed to the options listed in Section 11.1 which must be used **before** the first call to nag_quad_1d_gen_vec_multi_rcomm (d01rac).

Index LDI i query only

I_{ldi}, the index of **icom** required for obtaining ldi. See Section 9.1.

Index LDR i query only

I_{ldr}, the index of **icom** required for obtaining ldr. See Section 9.1.

Index NSDIV i query only

I_{nsdiv}, the index of **icom** required for obtaining $nsdiv$. See Section 9.1.

Index NSEG i query only

I_{nseg}, the index of **icom** required for obtaining $nseg$. See Section 9.1.

Index FCP i query only

I_{fcp}, the index of **icom** required for obtaining fcp. See Section 9.1.

Index EVALSP i query only

I_{evalsp}, the index of **icom** required for obtaining $evalsp$. See Section 9.1.

Index DSP i query only

I_{dsp}, the index of **icom** required for obtaining dsp. See Section 9.1.

Index ESP i query only

I_{esp}, the index of **icom** required for obtaining esp. See Section 9.1.

Index SINFOIP i query only

$I_{sinfoip}$, the index of **icom** required for obtaining $sinfoip$. See Section 9.1.

Index SINFORP i query only

$I_{sinforp}$, the index of **icom** required for obtaining $sinforp$. See Section 9.1.