NAG Toolbox

nag_quad_1d_gen_vec_multi_rcomm (d01ra)

1 Purpose

nag_quad_1d_gen_vec_multi_rcomm (d01ra) is a general purpose adaptive integrator which calculates an approximation to a vector of definite integrals **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 (d01ra) is particularly effective for the integration of a vector of similar functions.

2 Syntax

```
[irevcm, sid, needi, x, nx, dinest, errest, icom, com, ifail] =
nag_quad_1d_gen_vec_multi_rcomm(irevcm, a, b, needi, x, nx, fm, dinest, errest,
iopts, opts, icom, com, 'ni', ni, 'lenx', lenx)
[irevcm, sid, needi, x, nx, dinest, errest, icom, com, ifail] = d01ra(irevcm, a,
b, needi, x, nx, fm, dinest, errest, iopts, opts, icom, com, 'ni', ni, 'lenx',
lenx)
```

3 Description

nag_quad_1d_gen_vec_multi_rcomm (d01ra) 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 (d01ra) 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 (d01ra) is the integration function in the suite of functions nag_quad_1d_gen_vec_multi_rcomm (d01ra) and nag_quad_1d_gen_vec_multi_dimreq (d01rc). It also uses optional parameters, which can be set and queried using the functions nag_quad_opt_set (d01zk) and nag_quad_opt_get (d01zl) respectively. The options available are described in Section 11.

First, the option arrays **iopts** and **opts** must be initialized using nag_quad_opt_set (d01zk). Thereafter any required options must be set before calling nag_quad_1d_gen_vec_multi_rcomm (d01ra), or the function nag_quad_1d_gen_vec_multi_dimreq (d01rc).

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

Setup phase

```
iopts = zeros(100, 1, nag_int_name);
 opts  = zeros(100, 1);

 % initialize option arrays
 [iopts, opts, ifail] = d01zk('Initialize = d01ra', iopts, opts);

 % set any non-default options required
 [iopts, opts, ifail] = d01zk('option = value', iopts, opts);
 ...
```

```
% determine maximum required array lengths
[lenxrq, ldfmrq, sdfmrq, licmin, licmax, lcmin, lcmax, ifail] = ...
      d01rc(ni, iopts, opts);

% allocate remaining arrays
needi  = zeros(ni, 1, nag_int_name);
com    = zeros(lcmax, 1);
icom   = zeros(licmax, 1, nag_int_name);
fm     = zeros(ldfmrq, sdfmrq);
dinest = zeros(ni, 1);
errest = zeros(ni, 1);
x      = zeros(1, lenxrq);
```

Solve phase

```
irevcm = nag_int(1);

while irevcm ~= 0
  [irevcm, sid, needi, x, nx, dinest, errest, icomm, comm, ifail] = ...
    d01ra(irevcm, a, b, needi, x, nx, fm, dinest, errest, ...
          iopts, opts, icom, com);

  switch 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)
        Final return
  end
```

Diagnostic phase

```
[ivalue, rvalue, cvalue, optype, ifail] = d01zl('option', iopts, opts);
```

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) on completion of nag_quad_1d_gen_vec_multi_rcomm (d01ra). Otherwise the direct solution will be returned in **dinest**(j). This is indicated by the value of **needi**(j) 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 Parameters

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**.

5.1 Compulsory Input Parameters

1: **irevcm** – INTEGER

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 re-entry: **irevcm** should normally be left unchanged. However, if **irevcm** is set to a negative value, nag_quad_1d_gen_vec_multi_rcomm (d01ra) will terminate, (see **irevcm** $= 11$ and **irevcm** $= 12$ above).

2: **a** – REAL (KIND=nag_wp)

a, the lower bound of the domain.

3: **b** – REAL (KIND=nag_wp)

b, the upper bound of the domain.

If $|b - a| < 10\epsilon$, where ϵ is the ***machine precision*** (see nag_machine_precision (x02aj)), then nag_quad_1d_gen_vec_multi_rcomm (d01ra) will return **dinest**$(j) =$ **errest**$(j) = 0.0$, for $j = 1, 2, \ldots, n_i$.

4: **needi**(**ni**) – INTEGER array

On initial entry: need not be set.

On intermediate re-entry: **needi**(j) may be used to indicate what action you have taken for integral j.

needi$(j) = 1$
You have provided the values $f_j(x_i)$ in **fm**(j, i), for $i = 1, 2, \ldots, n_x$.

needi$(j) < 0$
You are abandoning the evaluation of integral j. The current values of **dinest**(j) and **errest**(j) will be returned on final completion.

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

5: **x**(**lenx**) – REAL (KIND=nag_wp) array

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, $\mathbf{x}(i)$ must contain a break-point x_i^0, for $i = 1, 2, \ldots, \left(s_{pri} - 1\right)$, 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 $i = 1, 2, \ldots, \left(s_{pri} - 1\right)$.

6: **nx** – INTEGER

On initial entry: need not be set.

On intermediate re-entry: must not be changed.

7: **fm**$(ldfm, :)$ – REAL (KIND=nag_wp) array

The first dimension of the array **fm** must be at least $ldfmrq$, 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 (d01rc). If default options are chosen, $ldfmrq = n_i$, implying $ldfm \geq$ **ni**.

The second dimension of the array **fm** must be at least $sdfmrq$, 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 (d01rc). If default options are chosen, $sdfmrq = lenxrq$.

On initial entry: need not be set.

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

8: **dinest**(**ni**) – REAL (KIND=nag_wp) array

dinest(j) 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.

9: **errest**(**ni**) – REAL (KIND=nag_wp) array

errest(j) 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.

10: **iopts**(**100**) – INTEGER array

11: **opts**(**100**) – REAL (KIND=nag_wp) array

The arrays **iopts** and **opts must not** be altered between calls to any of the functions nag_quad_1d_gen_vec_multi_rcomm (d01ra), nag_quad_1d_gen_vec_multi_dimreq (d01rc), nag_quad_opt_set (d01zk) and nag_quad_opt_get (d01zl).

12: **icom**$(licom)$ – INTEGER array

$licom$, the dimension of the array, must satisfy the 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 (d01rc). 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..

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.

13: **com**$(lcom)$ – REAL (KIND=nag_wp) array

$lcom$, the dimension of the array, must satisfy the 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 (d01rc). 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 (d01rc). If default options are chosen, $lcmax = 94 + 9 \times$ **ni** $+ \lceil$**ni**$/2\rceil + \left(s_{pri} + 100\right) \times \left(2 + \lceil\right.$**ni**$\left./2\right\rceil + 2 \times$ **ni**)..

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.

5.2 Optional Input Parameters

1: **ni** – INTEGER

Default: the dimension of the arrays **needi**, **dinest**, **errest** and the first dimension of the array **fm**. (An error is raised if these dimensions are not equal.)

n_i, the number of integrands.

2: **lenx** – INTEGER

Default: the dimension of the array **x**.

The dimension of the array **x**. currently **lenx** $= \max\left(122, s_{pri} - 1\right)$ 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 (d01rc).

5.3 Output Parameters

1: **irevcm** – INTEGER

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

irevcm requests the integrands $f_j(x_i)$ be evaluated for all required $j \in 1, \ldots, n_i$ as indicated by **needi**, and at all the points x_i, for $i = 1, 2, \ldots, n_x$. Abscissae x_i are provided in **x**(i) 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**$(j) = 1$, $f_j(x_i)$ 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**$(j) = 0$, 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), for $j = 1, 2, \ldots, n_i$, will be set to this negative value and **ifail** $= -1$ will be returned.

During the adaptive solve phase:

irevcm $= 12$
Function values are required to improve the estimates of the definite integrals.

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

If **needi**$(j) = 1$, $f_j(x_i)$ must be provided in **fm**(j, i).

If **needi**$(j) = 2$, 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) 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 **ifail** $= 1$, 2 or 3 will be returned and the elements of **needi** will reflect the current state of the adaptive process.

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

irevcm $= 0$
Indicates the algorithm has completed.

2: **sid** – INTEGER

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 (d01ra) return the same value of **sid**, i.e., the same set of abscissae was used.

3: **needi**(**ni**) – INTEGER array

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

needi$(j) = 0$
Do not provide $f_j(x_i)$. Any provided values will be ignored.

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

needi$(j) = 2$
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**$(j) = 1$, and supply $f_j(x_i)$ in **fm**(j, i), for $i = 1, 2, \ldots, n_x$.

needi$(j) = 3$
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**$(j) = 1$, and supply $f_j(x_i)$ in **fm**(j, i), for $i = 1, 2, \ldots, n_x$.

needi$(j) = 4$
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) 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**$(j) = 1$. Integral and error estimation will be performed again during the next iteration.

On final exit: **needi**(j) indicates the final state of integral j.

needi$(j) = 0$
The error estimate for F_j is below the requested tolerance.

needi$(j) = 1$
The error estimate for F_j is below the requested tolerance after extrapolation.

needi$(j) = 2$
The error estimate for F_j is above the requested tolerance.

needi$(j) = 3$
The error estimate for F_j is above the requested tolerance, and extremely bad behaviour of integral j has been detected.

needi$(j) < 0$
You prohibited further evaluation of integral j.

4: **x**(**lenx**) – REAL (KIND=nag_wp) array

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

5: **nx** – INTEGER

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

6: **dinest**(**ni**) – REAL (KIND=nag_wp) array

dinest(j) contains the current estimate of the definite integral F_j.

Contains the current estimates of the **ni** integrals. If **irevcm** $= 0$, this will be the final solution.

7: **errest**(**ni**) – REAL (KIND=nag_wp) array

errest(j) contains the current error estimate of the definite integral F_j.

Contains the current error estimates for the **ni** integrals. If **irevcm** $= 0$, **errest** contains the final error estimates of the **ni** integrals.

8: **icom**$(licom)$ – INTEGER array

9: **com**$(lcom)$ – REAL (KIND=nag_wp) array

10: **ifail** – INTEGER

On final exit: **ifail** $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Note: nag_quad_1d_gen_vec_multi_rcomm (d01ra) may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the function:

ifail $= 1$ (*warning*)

At least one error estimate exceeded the requested tolerances.

ifail $= 2$ (*warning*)

Extremely bad behaviour was detected for at least one integral.

ifail $= 3$ (*warning*)

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

ifail $= 11$

irevcm had an illegal value.

ifail $= 21$

Constraint: **ni** ≥ 1.

ifail $= 71$

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

ifail $= 81$

ifail $= 111$

ifail $= 171$ (*warning*)

ifail $= 191$ (*warning*)

ifail $= 1001$

Either the option arrays **iopts** and **opts** have not been initialized for nag_quad_1d_gen_vec_multi_rcomm (d01ra), or they have become corrupted.

ifail $= 1101$

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

ifail $= -1$ (*warning*)

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

ifail $= -99$

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

ifail $= -399$

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

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

nag_quad_1d_gen_vec_multi_rcomm (d01ra) cannot guarantee, but in practice usually achieves, the following accuracy for each integral F_j:

$$\left|F_j - \mathbf{dinest}(j)\right| \leq \text{tol}$$

where

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

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,

$$\left|F_j - \mathbf{dinest}(j)\right| \leq \mathbf{errest}(j) \leq \text{tol}.$$

8 Further Comments

The time required by nag_quad_1d_gen_vec_multi_rcomm (d01ra) is usually dominated by the time required to evaluate the values of the integrands f_j.

nag_quad_1d_gen_vec_multi_rcomm (d01ra) 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 (d01ra).

nag_quad_1d_gen_vec_multi_rcomm (d01ra) 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.

8.1 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 (d01ra) with **ifail** $= 0$, 1, 2 or 3, 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**.

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

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

nsdiv The number of segments that have been subdivided during the adaptive process.
$nsdiv =$ **icom**(I_{nsdiv}).

nseg The total number of segments formed.
$nseg = 2nsdiv + s_{pri}$.
$nseg =$ **icom**(I_{nseg}).

dsp The reference of the first element of the array ds stored in **com**.
$dsp =$ **icom**(I_{dsp}).

esp The reference of the first element of the array es stored in **com**.
$esp =$ **icom**(I_{esp}).

evalsp The reference of the first element of the array $evals$ stored in **icom**.
$evalsp =$ **icom**(I_{evalsp}).

fcp The reference of the first element of the array $fcount$ stored in **icom**.
$fcp =$ **icom**(I_{fcp}).

sinforp The reference of the first element of the array $sinfor$ stored in **com**.
$sinforp =$ **icom**$(I_{sinforp})$.

sinfoip The reference of the first element of the array $sinfoi$ stored in **icom**.
$sinfoip =$ **icom**$(I_{sinfoip})$.

One-dimensional arrays:

$fcount(n_i)$
$fcount(1) =$ **icom**(fcp).

$fcount(j)$ contains the number of different approximations of integral j calculated, for $j = 1, 2, \ldots, n_i$.

Two-dimensional arrays:

$sinfoi(5, nseg)$
$sinfoi(1, 1) =$ **icom**$(sinfoip)$.
$sinfoi$ contains information about the hierarchy of splitting.

$sinfoi(1, k)$ contains the split identifier for segment k, for $k = 1, 2, \ldots, nseg$.

$sinfoi(2, k)$ contains the parent segment number of segment k (i.e., the segment was split to create segment k), for $k = 1, 2, \ldots, nseg$.

$sinfoi(3, k)$ and $sinfoi(4, k)$ 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.

$sinfoi(5, k)$ 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 $k = 1, 2, \ldots, nseg$. A negative level indicates that segment k will not be split further, the level is then given by the absolute value of $sinfoi(5, k)$.

$sinfor(2, nseg)$
$sinfor(1, 1) =$ **com**$(sinforp)$.
$sinfor$ contains the bounds of each segment.

$sinfor(1, k)$ contains the lower bound of segment k, for $k = 1, 2, \ldots, nseg$.

$sinfor(2, k)$ contains the upper bound of segment k, for $k = 1, 2, \ldots, nseg$.

$evals(n_i, nseg)$
$evals(1, 1) =$ **icom**$(evalsp)$.

$evals$ 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 $j = 1, 2, \ldots, n_i$ and $k = 1, 2, \ldots, nseg$.

$evals(j, k) = 0$ indicates that integral j has not been evaluated over segment k.

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

$evals(j, k) = 2$ 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 **needi**$(j) < 0$.

$evals(j, k) = 3$ indicates that integral j has been evaluated over segment k, and this evaluation no longer contributes to the direct estimate of F_j.

$evals(j, k) = 4$ 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 (d01ra) to return **ifail** $= 2$ or 3, indicating extremely bad behaviour.

$evals(j, k) = 5$ 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.

$ds(n_i, nseg)$
$ds(1, 1) =$ **com**(dsp).

$ds(j, k)$ 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 $k = 1, 2, \ldots, nseg$.

$es(n_i, nseg)$
$es(1,1) = \mathbf{com}(esp)$.

$es(j,k)$ 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, $K_j = \{k \mid evals(j,k) = 1, 2, 4 \text{ or } 5,\ 1 \le k \le nseg\}$. D_j will have been returned in **dinest**(j), unless extrapolation was successful, as indicated by **needi**(j).

Similarly, E_j will have been returned in **errest**(j) 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.

9 Example

This example integrates

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

9.1 Program Text

```
function d01ra_example

fprintf('d01ra example results\n\n');

  % Setup phase.

  % set problem parameters
  ni = nag_int(2);
  nx = nag_int(0);
  % lower (a) and upper (b) bounds
  a = 0;
  b = pi;
  iopts = zeros(100, 1, nag_int_name);
  opts  = zeros(100, 1);

  % initialize option arrays
  [iopts, opts, ifail] = d01zk('Initialize = d01ra', iopts, opts);

  % set any non-default options required
  [iopts, opts, ifail] = d01zk('Quadrature Rule = gk41', iopts, opts);
  [iopts, opts, ifail] = d01zk('Absolute Tolerance = 1.0e-7', iopts, opts);
  [iopts, opts, ifail] = d01zk('Relative Tolerance = 1.0e-7', iopts, opts);

  % determine maximum required array lengths
  [lenxrq, ldfmrq, sdfmrq, licmin, licmax, lcmin, lcmax, ifail] = ...
        d01rc(ni, iopts, opts);

  % allocate remaining arrays
  needi  = zeros(ni, 1, nag_int_name);
```

```
comm   = zeros(lcmax, 1);
icomm  = zeros(licmax, 1, nag_int_name);
fm     = zeros(ldfmrq, sdfmrq);
dinest = zeros(ni, 1);
errest = zeros(ni, 1);
x      = zeros(1, lenxrq);

% Solve phase.

% Use d01ra to evaluate the definate integrals of:
%   f_1 = (x*sin(2*x))*cos(15*x)
%   f_2 = (x*sin(2*x))*(x*cos(50*x))

% set initial irevcm
irevcm = nag_int(1);

while irevcm ~= 0
  [irevcm, sid, needi, x, nx, dinest, errest, icomm, comm, ifail] = ...
    d01ra(irevcm, a, b, needi, x, nx, fm, dinest, errest, ...
          iopts, opts, icomm, comm);

  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.

      % Calculate x*sin(2*x), storing the result in fm(2,1:nx) for re-use.
      fm(2, :) = x.*sin(2*x);

      % Calculate f_1
      fm(1, :) = fm(2, :).*cos(15*x);

      % Calculate f_2
      fm(2, :) = fm(2, :).*x.*cos(50*x);
    case 12
      % Intermediate returns.
      % These will occur during the adaptive phase.
      % All requested values must be supplied.
      % dinest and errest do not contain approximations
      % over the complete interval at this stage.

      % Calculate x*sin(2*x).
      fm(2, :) = x.*sin(2*x);

      % Calculate f_1 if required
      if needi(1) == 1
        fm(1, :) = fm(2, :).*cos(15*x);
      end

      % Complete f_2 calculation if required.
      if needi(2) == 1
        fm(2, :) = fm(2, :).*x.*cos(50*x);
      end
    case 0
      % Final return
  end
end

% query some currently set options and statistics.
[ivalue, rvalue, cvalue, optype, ifail] = ...
   d01zl('Quadrature rule', iopts, opts);
display_option('Quadrature rule',optype,ivalue,rvalue,cvalue);
[ivalue, rvalue, cvalue, optype, ifail] = ...
   d01zl('Maximum Subdivisions', iopts, opts);
display_option('Maximum Subdivisions',optype,ivalue,rvalue,cvalue);
[ivalue, rvalue, cvalue, optype, ifail] = ...
   d01zl('Extrapolation', iopts, opts);
display_option('Extrapolation',optype,ivalue,rvalue,cvalue);
```

```
  [ivalue, rvalue, cvalue, optype, ifail] = ...
     d01zl('Extrapolation Safeguard', iopts, opts);
  display_option('Extrapolation Safeguard',optype,ivalue,rvalue,cvalue);

  % print solution
  fprintf('\nIntegral |  needi  |   dinest   |   errest   \n');
  for j=1:ni
    fprintf('%9d %9d %12.4e %12.4e\n', j, needi(j), dinest(j), errest(j));
  end

function [dinest, errest, user] = monit(ni, ns, dinest, errest, fcount, ...
                                        sinfoi, evals, ldi, sinfor, fs, ...
                                        es, ldr, user)
  % Display information on individual segments
  fprintf('\nInformation on splitting and evaluations over subregions.\n');
  for k=1:ns
    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);
    fprintf('\nSegment %3d Sid = %3d', k, sid);
    fprintf(' Parent = %3d Level = %3d.\n', parent, level);
    if (child1>0)
      fprintf('Children = (%3d, %3d)\n', child1, child2);
    end
    fprintf('Bounds (%11.4e, %11.4e)\n', lbnd, ubnd);
    for j = 1:ni
      if (evals(j,k) ~= 0)
        fprintf('Integral %2d approximation %11.4e\n', j, fs(j,k));
        fprintf('Integral %2d error estimate %11.4e\n', j, es(j,k));
      end
      if (evals(j,k) ~= 1)
        fprintf('Integral %2d evaluation', j);
        fprintf(' has been superseded by descendants.\n');
      end
    end
  end

function display_option(optstr,optype,ivalue,rvalue,cvalue)
  % Query optype and print the appropriate option values

  switch optype
    case 1
      fprintf('%30s: %13d\n', optstr, ivalue);
    case 2
      fprintf('%30s: %13.4e\n', optstr, rvalue);
    case 3
      fprintf('%30s: %16s\n', optstr, cvalue);
    case 4
      fprintf('%30s: %3d  %16s\n', optstr, ivalue, cvalue);
    case 5
      fprintf('%30s: %14.4e  %16s\n', optstr, rvalue, cvalue);
  end
```

9.2 Program Results

```
     d01ra example 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
```

10 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

10.1 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 symbol represents various machine constants:

ϵ represents the ***machine precision*** (see nag_machine_precision (x02aj)).

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 (d01zl). They will have no effect if passed to nag_quad_opt_set (d01zk).

For nag_quad_1d_gen_vec_multi_rcomm (d01ra) the maximum length of the argument **cvalue** used by nag_quad_opt_get (d01zl) 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 (d01ra) will automatically generate s_{pri} segments of equal size covering the interval $[a, b]$.

MANUAL
: nag_quad_1d_gen_vec_multi_rcomm (d01ra) 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_quad_1d_fin_bad_vec (d01at)

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.

10.2 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 (d01ra) 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 (d01ra).

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.