# NAG Library Routine Document

# D01RAF

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

**Note**: *this routine 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 9 of this document. If, however, you wish to reset some or all of the settings please refer to Section 10 for a detailed description of the specification of the optional parameters.*

## 1 Purpose

D01RAF 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'. D01RAF is particularly effective for the integration of a vector of similar functions.

## 2 Specification

```
SUBROUTINE D01RAF (IREVCM, NI, A, B, SID, NEEDI, X, LENX, NX, FM, LDFM,         &
                   DINEST, ERREST, IOPTS, OPTS, ICOMM, LICOMM, COMM, LCOMM,     &
                   IFAIL)

INTEGER            IREVCM, NI, SID, NEEDI(NI), LENX, NX, LDFM, IOPTS(100),      &
                   ICOMM(LICOMM), LICOMM, LCOMM, IFAIL
REAL (KIND=nag_wp) A, B, X(LENX), FM(LDFM,*), DINEST(NI), ERREST(NI),           &
                   OPTS(100), COMM(LCOMM)
```

## 3 Description

D01RAF is an extension to various QUADPACK routines, including QAG, QAGS and QAWSE. The extensions made allow multiple integrands to be evaluated simultaneously, using a vectorized interface and reverse communication.

The quadrature scheme employed by D01RAF 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 $\epsilon$-algorithm (see Wynn (1956)) to perform extrapolation. The local error estimation is described in Piessens *et al.* (1983).

D01RAF is the integration routine in the suite of routines D01RAF, D01RBF, D01RCF and D01ZKF.

First, the option arrays IOPTS and OPTS must be initialized using D01ZKF. Thereafter any required options must be set before calling D01RAF, or the routines D01RBF and D01RCF. The options available are described in Section 10.

A typical usage of this suite of routines is (in psuedo-code for clarity),

*Setup phase*

```
liopts = 100
lopts = 100
allocate(iopts(liopts),opts(lopts))
call D01ZKF('initialize = d01raf',iopts,liopts,opts,lopts,
     ifail)
call D01ZKF('option = value',iopts,liopts,opts,lopts,
     ifail)
...
```

```
call D01RCF(ni,lenxrq,ldfmrq,sdfmrq,licmin,licmax,lcmin,lcmax,iopts,opts,
     ifail)
lenx = lenxrq
ldfm = ldfmrq
sdfm = sdfmrq
licomm = licmax
lcomm = lcmax
allocate(icomm(licomm),comm(lcomm),x(lenx),fm(ldfm,sdfm),needi(ni),
              dinest(ni),errest(ni))
```

*Solve phase*

```
irevcm = 1
while irevcm≠0
    call D01RAF(irevcm,ni,a,b,sid,needi,x,lenx,nx,fm,ldfm,   &
                dinest,errest,iopts,opts,icomm,licomm,comm, &
                lcomm,ifail)
    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 ifail
    end select
end while
```

*Diagnostic phase*

```
call D01ZLF('option',ivalue,rvalue,cvalue,optype,iopts,opts,ifail)
...
call D01RBF(monit,ni,dinest,errest,iopts,opts,icomm,licomm,comm,lcomm, &
                 iuser,ruser,ifail)
```

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 breakpoints 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 routine enters the adaptive phase. The estimated errors are tested against the requested tolerances $\epsilon_a$ and $\epsilon_r$, 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 favour segments with a lower level over segments with a high level and potentially worse error estimate or not. Up to $\max_{sdiv}$ subdivisions are allowed provided sufficient memory, where $\max_{sdiv}$ is the value of **Maximum Subdivisions**.

Once a sufficient number of error estimates have been constructed for a particular integral, the routine 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 $\epsilon_{safe}$ can be set using **Extrapolation Safeguard**, and the extrapolated solution will only be considered if $\epsilon_{safe}\epsilon_q \leq \epsilon_{ex}$, where $\epsilon_q$ and $\epsilon_{ex}$ are the estimated error directly from the quadrature and from the extrapolation respectively.

## 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, Überhuber 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 routine uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the parameter **IREVCM**. Between intermediate exits and re-entries, **all parameters other than IREVCM, NEEDI and FM must remain unchanged**.

1: IREVCM – INTEGER *Input/Output*

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

IREVCM $= 1$
Set up data structures in ICOMM and COMM and start a new integration.

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

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

IREVCM requests the values of $f_j(x_i)$ 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$(i)$ 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 inital 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 $j = 1, 2, \dots, n_i$, 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 returned.

If NEEDI$(j) = 2$, 3 or 4, the current error estimate of integral $j$ does not require integrand $j$ to be evaluated. 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 either IFAIL $= 1$, 2 or 3 will be returned and the elements of NEEDI will reflect the current state of of the adaptive process.

*On intermediate re-entry*: IREVCM should normally be left unchanged. However, if IREVCM is set to a negative value, D01RAF 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*: $n_i$, number of integrands.

3: A – REAL (KIND=nag_wp) *Input*

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

4: B – REAL (KIND=nag_wp) *Input*

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

If $|b - a| < 10\epsilon$, where $\epsilon$ is the ***machine precision*** (see X02AJF), then D01RAF will return DINEST$(j) =$ ERREST$(j) = 0.0$, for $j = 1, 2, \ldots, n_i$.

5: SID – INTEGER *Output*

*For advanced users.*

*On intermediate exit*: SID identifies a specific set of abscissae, $\mathbf{x}$, 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 $\mathbf{x}$, and reuse them should D01RAF return the same value of SID, i.e., the same set of abscissae was used.

6: NEEDI(NI) – INTEGER array *Input/Output*

*On initial entry*: need not be set.

*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 definitely 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 integral 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 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 $f_j(x_i)$.

*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 $j$.

7: X(LENX) – REAL (KIND=nag_wp) array *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 may be used to supply a set of initial 'break points' inside the domain of integration. Specifically, X$(i)$ 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, $x_i^0 \in (a, b)$, $|x_i^0 - a| > 10.0\epsilon$, $|x_i^0 - b| > 10.0\epsilon$, for $i = 1, 2, \ldots, (s_{pri} - 1)$.

*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.

8: LENX – INTEGER *Input*

*On entry*: the dimension of the array X as declared in the (sub)program from which D01RAF is called. 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 10). $lenxrq$ is returned as LENXRQ from D01RCF.

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(LDFM,$*$) – REAL (KIND=nag_wp) array *Input*

**Note**: 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 D01RCF. 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 $j = 1, 2, \ldots, n_i$.

11: LDFM – INTEGER *Input*

*On entry*: the first dimension of the array FM as declared in the (sub)program from which D01RAF is called.

*Constraint*: LDFM $\geq ldfmrq$, where $ldfmrq$ is dependent upon $n_i$ and the options currently set. $ldfmrq$ is returned as LDFMRQ from D01RCF. If default options are chosen, $ldfmrq = n_i$, implying LDFM $\geq$ NI.

12: DINEST(NI) – REAL (KIND=nag_wp) array *Input/Output*

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.

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

13: ERREST(NI) – REAL (KIND=nag_wp) array *Input/Output*

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.

*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) – INTEGER array *Input*

*On entry*: integer option array generated by D01ZKF.

*Constraint*: IOPTS must not be altered between calls to D01RAF, D01RCF, D01ZKF and D01ZLF.

15: OPTS(100) – REAL (KIND=nag_wp) array *Input*

*On entry*: real option array generated by D01ZKF.

*Constraint*: OPTS must not be altered between calls to D01RAF, D01RCF, D01ZKF and D01ZLF.

16: ICOMM(LICOMM) – INTEGER array *Communication Array*

The contents of this array **must not** be changed directly once the integration process has started. ICOMM 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. You are free to examine the contents of ICOMM using the diagnostic routine D01RBF at any time during, or after, the integration process.

17: LICOMM – INTEGER *Input*

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

*Constraint*: LICOMM $\geq licmin$, where $licmin$ is dependent upon NI and the current options set. $licmin$ is returned as LICMIN from D01RCF. If the default options are set, then

$licmin = 55 + 6 \times \mathrm{NI}$. 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 D01RCF. If default options are chosen, except for possibly increasing the value of $s_{pri}$, then $licmax = 50 + 5 \times \mathrm{NI} + (s_{pri} + 100) \times (5 + \mathrm{NI})$.

18: COMM(LCOMM) – REAL (KIND=nag_wp) array *Communication Array*

The contents of this array **must not** be changed directly once the integration process has started. COMM 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. You are free to examine the contents of COMM using the diagnostic routine D01RBF at any time during, or after, the solution phase.

19: LCOMM – INTEGER *Input*

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

*Constraint*: LCOMM $> lcmin$, where $lcmin$ is dependent upon NI, $s_{pri}$ and the current options set. $lcmin$ is returned as LCMIN from D01RCF. If default options are set, then $lcmin = 96 + 12 \times \mathrm{NI}$. Larger values are recommended if you anticipate that any integrals will require the domain to be further subdivided.

Given the current options and parameters, the maximum value, $lcmax$, of LCOMM that may be required, is returned as LCMAX from D01RCF. If default options are chosen, $lcmax = 94 + 9 \times \mathrm{NI} + \lceil \mathrm{NI}/2 \rceil + (s_{pri} + 100) \times (2 + \lceil \mathrm{NI}/2 \rceil + 2 \times \mathrm{NI})$.

20: IFAIL – INTEGER *Input/Output*

*On initial entry*: IFAIL must be set to 0, $-1$ or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value $-1$ or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, because for this routine the values of the output parameters may be useful even if IFAIL $\neq 0$ on exit, the recommended value is $-1$. **When the value $-1$ or 1 is used it is essential to test the value of IFAIL on exit.**

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

## 6 Error Indicators and Warnings

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

**Note**: D01RAF may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the routine:

IFAIL $= 1$

At least one error estimate exceeded the requested tolerances.

IFAIL $= 2$

Extremely bad behaviour was detected for at least one integral.

IFAIL $= 3$

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.
On entry, IREVCM $= \langle value \rangle$.

IFAIL $= 21$

On entry, NI $= \langle value \rangle$.
Constraint: NI $\geq 1$.

IFAIL $= 71$

On entry, **Primary Division Mode** $=$ MANUAL and at least one supplied breakpoint in X is outside of the domain of integration.

IFAIL $= 81$

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

IFAIL $= 111$

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

IFAIL $= 171$

The length of LICOMM is insufficient for additional subdivision.
On entry, LICOMM $= \langle value \rangle$.
Constraint: LICOMM $\geq \langle value \rangle$.

IFAIL $= 191$

The length of LCOMM is insufficient for additional subdivision.
On entry, LCOMM $= \langle value \rangle$.
Constraint: LCOMM $\geq \langle value \rangle$.

IFAIL $= 1001$

Either the option arrays IOPTS and OPTS have not been initialized for D01RAF, or they have become corrupted.

IFAIL $= 1101$

On entry, one of ICOMM and COMM has become corrupted.

IFAIL $= -1$

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

## 7 Accuracy

D01RAF cannot guarantee, but in practice usually achieves, the following accuracy for each integral $F_j$:

$$\left|F_j - \mathrm{DINEST}(j)\right| \leq \mathrm{tol}$$

where

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

$\epsilon_a$ and $\epsilon_r$ are the error tolerances **Absolute Tolerance** and **Relative Tolerance** respectively. Moreover, it returns ERREST, the entries of which in normal circumstances satisfy,

$$\left|F_j - \text{DINEST}(j)\right| \leq \text{ERREST}(j) \leq \text{tol}.$$

## 8 Further Comments

The time required by D01RAF is usually dominated by the time required to evaluate the values of the integrands $f_j$.

D01RAF 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 D01RAF.

The diagnostic routine D01RBF may be used to examine the subdivision strategy used by D01RAF both during intermediate stages and after completion. In particular it may be used to examine how individual integrals behaved over individual sub-intervals, and so may be used to determine where integrals behaved extremely badly should D01RAF return with IFAIL $= 2$ or 3.

D01RAF 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 $\delta_a$ and $\delta_r$ can be set through the optional parameters **Absolute Interval Minimum** and **Relative Interval Minimum** respectively. You may try using D01RAF again over such sub-intervals with only the integrals that exhibited bad behaviour and a new set of communication arrays, or you may use another quadrature routine designed to deal with any known specific singular behaviour over this sub-interval. If these provide sufficient accuracy over the specific sub-intervals, you may correct the definite integral evaluations over $[a, b]$ by subtracting the inadequate sub-interval estimations from DINEST and ERREST and adding the acceptable sub-interval estimations generated by the direct integration of the sub-interval where bad behaviour was detected. See Section 8 in D01RBF.

## 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

```
!   Mark 24 Release.  NAG Copyright 2012.
    Module d01rafe_mod

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                   :: nout = 6
      Logical, Parameter                   :: disp_split_info = .True.
    Contains
      Subroutine d01rbf_monit(ni,ns,defint,errest,fcount,sinfoi,evals,ldi, &
        sinfor,fs,es,ldr,iuser,ruser)

!       .. Scalar Arguments ..
        Integer, Intent (In)                 :: ldi, ldr, ni, ns
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout)   :: defint(ni), errest(ni), ruser(*)
        Real (Kind=nag_wp), Intent (In)      :: es(ldr,*), fs(ldr,*),          &
                                                sinfor(ldr,*)
        Integer, Intent (In)                 :: evals(ldi,*), fcount(ni),      &
                                                sinfoi(ldi,*)
        Integer, Intent (Inout)              :: iuser(*)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                   :: lbnd, ubnd
        Integer                              :: child1, child2, j, k, level,   &
                                                parent, sid
!       .. Executable Statements ..
        Continue
!       Display information on individual segments.
        If (.Not. disp_split_info) Then
          Go To 100
        End If
        Write (nout,*)
        Write (nout,99993)
        Do k = 1, ns
          Write (nout,*)
          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)
          Write (nout,99999) k, sid, parent, level
          If (child1>0) Then
            Write (nout,99998) child1, child2
          End If
          Write (nout,99997) lbnd, ubnd
          Do j = 1, ni
            If (evals(j,k)/=0) Then
              Write (nout,99996) j, fs(j,k)
              Write (nout,99995) j, es(j,k)
              If (evals(j,k)/=1) Then
                Write (nout,99994) j
              End If
            End If
          End Do
        End Do
100     Continue
        Return
99999   Format (' Segment ',I3,', SID = ',I3,', Parent = ',I3,', Level = ',I3, &
          '.')
99998   Format (' Children = (',I3,',',I3,')')
99997   Format (' Bounds (',Es11.4,',',Es11.4,')')
99996   Format (' Integral ',I2,' approximation:',1X,Es11.4,'.')
99995   Format (' Integral ',I2,' error estimate:',1X,Es11.4,'.')
99994   Format (' Integral ',I2, &
          ' evaluation has been superseded by descendants.')
99993   Format (' Information on splitting and evaluations over subregions. ')

      End Subroutine d01rbf_monit
      Subroutine display_option(optstr,optype,ivalue,rvalue,cvalue)
!       Subroutine to query optype and print the appropriate option values

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: rvalue
        Integer, Intent (In)                 :: ivalue, optype
        Character (*), Intent (In)           :: cvalue, optstr
!       .. Executable Statements ..
        Select Case (optype)
        Case (1)
          Write (nout,99999) optstr, ivalue
        Case (2)
          Write (nout,99998) optstr, rvalue
        Case (3)
          Write (nout,99997) optstr, cvalue
        Case (4)
          Write (nout,99996) optstr, ivalue, cvalue
        Case (5)
          Write (nout,99995) optstr, rvalue, cvalue
        Case Default
        End Select

        Flush (nout)

        Return
99999   Format (3X,A30,' : ',I13)
99998   Format (3X,A30,' : ',Es13.4)
99997   Format (3X,A30,' : ',8X,A16)
99996   Format (3X,A30,' : ',I13,3X,A16)
99995   Format (3X,A30,' : ',Es13.4,3X,A16)
      End Subroutine display_option
    End Module d01rafe_mod

    Program d01rafe

!     .. Use Statements ..
      Use nag_library, Only: d01raf, d01rbf, d01rcf, d01zkf, d01zlf, x01aaf
      Use d01rafe_mod, Only: d01rbf_monit, display_option, nag_wp, nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                   :: a, b, pi, rvalue
      Integer                              :: ifail, irevcm, ivalue, j, lcmax, &
                                              lcmin, lcomm, lcusd, ldfm,       &
                                              ldfmrq, lenx, lenxrq, licmax,    &
                                              licmin, licomm, licusd, ni, nx,  &
                                              optype, sdfm, sdfmrq, sid
      Character (16)                       :: cvalue
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable      :: comm(:), defint(:), errest(:),   &
                                              fm(:,:), opts(:), ruser(:), x(:)
      Integer, Allocatable                 :: icomm(:), iopts(:), iuser(:),    &
                                              needi(:)
!     .. Intrinsic Procedures ..
      Intrinsic                            :: cos, sin
!     .. Executable Statements ..
      Continue
      Write (nout,*) 'D01RAF Example Program Results'
      Write (nout,*)

      pi = x01aaf(pi)

!     Setup phase.

!     set problem parameters
      ni = 2
!     lower (a) and upper (b) bounds
      a = 0.0E0_nag_wp
      b = pi
      Allocate (opts(100),iopts(100),iuser(1),ruser(1))

!     initialize option arrays
      ifail = 0
      Call d01zkf('Initialize = d01raf',iopts,100,opts,100,ifail)
!     set any non-default options required
      Call d01zkf('Quadrature Rule = gk41',iopts,100,opts,100,ifail)
      Call d01zkf('Absolute Tolerance = 1.0e-7',iopts,100,opts,100,ifail)
      Call d01zkf('Relative Tolerance = 1.0e-7',iopts,100,opts,100,ifail)

!     determine maximum requried array lengths
      ifail = -1
      Call d01rcf(ni,lenxrq,ldfmrq,sdfmrq,licmin,licmax,lcmin,lcmax,iopts, &
          opts,ifail)
        ldfm = ldfmrq
        sdfm = sdfmrq
        lenx = lenxrq
        licomm = licmax
        lcomm = lcmax

!       Allocate remaining arrays
        Allocate (icomm(licomm),needi(ni),comm(lcomm),fm(ldfm,sdfm),defint(ni), &
          errest(ni),x(lenx))


!         Solve phase.
!         Use D01RAF 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 = 1
        ifail = -1

        Do While (irevcm/=0)

          Call d01raf(irevcm,ni,a,b,sid,needi,x,lenx,nx,fm,ldfm,defint,errest, &
            iopts,opts,icomm,licomm,comm,lcomm,ifail)

          Select Case (irevcm)
          Case (11)
!               Initial returns.
!               These will occur during the non-adaptive phase.
!               All values must be supplied.
!               DEFINT 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(2,1:nx) = x(1:nx)*sin(2.0E0_nag_wp*x(1:nx))

!           Calculate f1
            fm(1,1:nx) = fm(2,1:nx)*cos(15.0E0_nag_wp*x(1:nx))

!           Complete f2 calculation.
            fm(2,1:nx) = fm(2,1:nx)*x(1:nx)*cos(50.0E0_nag_wp*x(1:nx))

          Case (12)
!               Intermediate returns.
!               These will occur during the adaptive phase.
!               All requested values must be supplied.
!               DEFINT and ERREST do not contain approximations
!                over the complete interval at this stage.
!
!               Calculate x*sin(2*x).
            fm(2,1:nx) = x(1:nx)*sin(2.0E0_nag_wp*x(1:nx))

!           Calculate f1 if required.
            If (needi(1)==1) Then
              fm(1,1:nx) = fm(2,1:nx)*cos(15.0E0_nag_wp*x(1:nx))
            End If

!           Complete f2 calculation if required.
            If (needi(2)==1) Then
              fm(2,1:nx) = fm(2,1:nx)*x(1:nx)*cos(50.0E0_nag_wp*x(1:nx))
            End If
          Case (0)
!           Final return. Test IFAIL.
            Select Case (ifail)
            Case (0:3)
!           Useful information has been returned.
            Case Default
!             An unrecoverable error has been detected.
              Go To 100
            End Select
          End Select

        End Do

!       query some currently set options and statistics.
        ifail = 0
        Call d01zlf('Quadrature rule',ivalue,rvalue,cvalue,optype,iopts,opts, &
          ifail)
        Call display_option('Quadrature rule',optype,ivalue,rvalue,cvalue)

        Call d01zlf('Maximum Subdivisions',ivalue,rvalue,cvalue,optype,iopts, &
          opts,ifail)
        Call display_option('Maximum Subdivisions',optype,ivalue,rvalue,cvalue)

        Call d01zlf('Extrapolation',ivalue,rvalue,cvalue,optype,iopts,opts, &
          ifail)
        Call display_option('Extrapolation',optype,ivalue,rvalue,cvalue)

        Call d01zlf('Extrapolation Safeguard',ivalue,rvalue,cvalue,optype,iopts, &
          opts,ifail)
        Call display_option('Extrapolation safeguard',optype,ivalue,rvalue, &
          cvalue)

!       print solution
        Write (nout,99999)
        Do j = 1, ni
          Write (nout,99998) j, needi(j), defint(j), errest(j)
        End Do

!       Investigate subdivision strategy
        Call d01rbf(d01rbf_monit,ni,defint,errest,icomm,licomm,licusd,comm, &
          lcomm,lcusd,iuser,ruser,ifail)

100     Continue

99999 Format (' Integral |  NEEDI   |   DEFINT   |   ERREST   ')
99998 Format (2(1X,I9),2(1X,Es12.4))
      End Program d01rafe
```

## 9.2 Program Data

None.

## 9.3 Program Results

```
 D01RAF Example Program Results

                  Quadrature rule :         GK41
             Maximum Subdivisions :            50
                    Extrapolation :         ON
          Extrapolation safeguard :    1.0000E-12
 Integral |  NEEDI   |   DEFINT   |   ERREST
         1         0  -2.8431E-02   1.1234E-14
         2         0   7.9083E-03   2.6600E-09

 Information on splitting and evaluations over subregions.

 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 =   1, 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 =   1, 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 =   2, 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 =   2, 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 =   3, 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 =   3, Parent =   2, Level =   3.
Bounds ( 7.8540E-01, 1.5708E+00)
Integral  2 approximation: -1.0365E-02.
Integral  2 error estimate:  1.7467E-10.
```

## 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 10.1.

Absolute Interval Minimum
Absolute Tolerance
Extrapolation
Extrapolation Safeguard
Maximum Subdivisions
Primary Division Mode
Primary Divisions
Prioritize Error
Quadrature Rule
Relative Interval Minimum
Relative Tolerance

### 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:

$\epsilon$ represents the ***machine precision***, typically $2^{-53}$ for 64-bit real numbers (see X02AJF).

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.

For D01RAF the maximum length of the parameter CVALUE used by D01ZLF 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 8.

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 $\epsilon$ algorithm (Wynn (1956)). **Extrapolation** can occasionally lead to premature convergence of the integrations.

ON
: Use extrapolation.

OFF
: Disable extrapolation.

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

$r = \epsilon_{safe}$. If $\epsilon_q$ is the estimated error from the quadrature evaluation alone, and $\epsilon_{ex}$ 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
: D01RAF will automatically generate $s_{pri}$ segments of equal size covering the interval $[a, b]$.

MANUAL
: D01RAF 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 $10.0\epsilon$ is automatically applied in all cases, and will result in fewer initial subdivisions being generated if automatically generated or supplied breakpoints 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 parameter 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 parameter 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 D01ATF

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 $= 1.0\text{E}{-6}$

$r = \delta_r$, 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 8.

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.