NAG Toolbox for MATLAB

d01ak

1 Purpose

d01ak is an adaptive integrator, especially suited to oscillating, nonsingular integrands, which calculates an approximation to the integral of a function $f(x)$ over a finite interval $[a,b]$:

$$I = \int_a^b f(x)\,dx.$$

2 Syntax

```
[result, abserr, w, iw, ifail] = d01ak(f, a, b, epsabs, epsrel, 'lw',
lw, 'liw', liw)
```

3 Description

d01ak is based on the QUADPACK routine QAG (see Piessens *et al.* 1983). It is an adaptive function, using the Gauss 30-point and Kronrod 61-point rules. A 'global' acceptance criterion (as defined by Malcolm and Simpson 1976) is used. The local error estimation is described in Piessens *et al.* 1983.

Because d01ak is based on integration rules of high order, it is especially suitable for nonsingular oscillating integrands.

d01ak requires you to supply a function to evaluate the integrand at a single point.

The function d01au uses an identical algorithm but requires you to supply a (sub)program to evaluate the integrand at an array of points. Therefore d01au will be more efficient if the evaluation can be performed in vector mode on a vector-processing machine.

d01au also has an additional parameter **key** which allows you to select from six different Gauss–Kronrod rules.

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

5 Parameters

5.1 Compulsory Input Parameters

1: **f – string containing name of m-file**

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

Its specification is:

```
[result] = f(x)
```

Input Parameters

1: **x – double scalar**

The point at which the integrand f must be evaluated.

Output Parameters

1: **result – double scalar**

The result of the function.

2: **a – double scalar**

a, the lower limit of integration.

3: **b – double scalar**

b, the upper limit of integration. It is not necessary that $a < b$.

4: **epsabs – double scalar**

The absolute accuracy required. If **epsabs** is negative, the absolute value is used. See Section 7.

5: **epsrel – double scalar**

The relative accuracy required. If **epsrel** is negative, the absolute value is used. See Section 7.

5.2 Optional Input Parameters

1: **lw – int32 scalar**

Default: The dimension of the array **w**.

The value of **lw** (together with that of **liw**) imposes a bound on the number of sub-intervals into which the interval of integration may be divided by the function. The number of sub-intervals cannot exceed **lw**/4. The more difficult the integrand, the larger **lw** should be.

Suggested value: **lw** = 800 to 2000 is adequate for most problems.

Default: 800

Constraint: **lw** ≥ 4.

2: **liw – int32 scalar**

Default: The dimension of the array **iw**.

The number of sub-intervals into which the interval of integration may be divided cannot exceed **liw**.

Suggested value: **liw** = **lw**/4.

Default: **lw**/4

Constraint: **liw** ≥ 1.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **result – double scalar**

The approximation to the integral I.

2: **abserr – double scalar**

An estimate of the modulus of the absolute error, which should be an upper bound for $|I - \textbf{result}|$.

3: **w(lw) – double array**

Details of the computation, as described in Section 8.

4: **iw(liw) – int32 array**

iw(1) contains the actual number of sub-intervals used. The rest of the array is used as workspace.

5: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail = 1

The maximum number of subdivisions allowed with the given workspace has been reached without the accuracy requirements being achieved. Look at the integrand in order to determine the integration difficulties. If necessary, another integrator, which is designed for handling the type of difficulty involved, must be used. Alternatively, consider relaxing the accuracy requirements specified by **epsabs** and **epsrel**, or increasing the amount of workspace.

ifail = 2

Round-off error prevents the requested tolerance from being achieved. Consider requesting less accuracy.

ifail = 3

Extremely bad local integrand behaviour causes a very strong subdivision around one (or more) points of the interval. The same advice applies as in the case of **ifail** = 1.

ifail = 4

On entry, **lw** < 4,
or **liw** < 1.

7 Accuracy

d01ak cannot guarantee, but in practice usually achieves, the following accuracy:

$$|I - \textbf{result}| \leq tol,$$

where

$$tol = \max\{|\textbf{epsabs}|, |\textbf{epsrel}| \times |I|\},$$

and **epsabs** and **epsrel** are user-specified absolute and relative error tolerances. Moreover, it returns the quantity **abserr** which, in normal circumstances, satisfies

$$|I - \textbf{result}| \leq \textbf{abserr} \leq tol.$$

8 Further Comments

The time taken by d01ak depends on the integrand and the accuracy required.

If **ifail** $\neq 0$ on exit, then you may wish to examine the contents of the array **w**, which contains the end points of the sub-intervals used by d01ak along with the integral contributions and error estimates over these sub-intervals.

Specifically, for $i = 1, 2, \ldots, n$, let r_i denote the approximation to the value of the integral over the sub-interval $[a_i, b_i]$ in the partition of $[a, b]$ and e_i be the corresponding absolute error estimate. Then, $\int_{a_i}^{b_i} f(x)\,dx \simeq r_i$ and **result** $= \sum_{i=1}^{n} r_i$. The value of n is returned in $\mathbf{iw}(1)$, and the values a_i, b_i, e_i and r_i are stored consecutively in the array **w**, that is:

$a_i = \mathbf{w}(i)$,

$b_i = \mathbf{w}(n + i)$,

$e_i = \mathbf{w}(2n + i)$ and

$r_i = \mathbf{w}(3n + i)$.

9 Example

```
d01ak_f.m

function [result] = d01ak_f(x)
        result=x*(sin(30.0*x))*cos(x);
```

```
a = 0;
b = 6.283185307179586;
epsabs = 0;
epsrel = 0.001;
[result, abserr, w, iw, ifail] = d01ak('d01ak_f', a, b, epsabs, epsrel)
```

```
result =
   -0.2097
abserr =
   4.4797e-14
w =
     array elided
iw =
     array elided
ifail =
           0
```