NAG Toolbox for MATLAB

d01ja

1 Purpose

d01ja attempts to evaluate an integral over an n-dimensional sphere ($n = 2$, 3, or 4), to a user-specified absolute or relative accuracy, by means of a modified Sag–Szekeres method. The function can handle singularities on the surface or at the centre of the sphere, and returns an error estimate.

2 Syntax

```
[result, esterr, nevals, ifail] = d01ja(f, ndim, radius, epsa, epsr, icoord, 'method', method)
```

3 Description

d01ja calculates an approximation to the n-dimensional integral

$$I = \int \cdots \int_S F(x_1, \ldots, x_n)\, dx_1 \cdots dx_n, \qquad 2 \leq n \leq 4,$$

where S is the hypersphere

$$\sqrt{(x_1^2 + \cdots + x_n^2)} \leq \alpha < \infty$$

(the integrand function may also be defined in spherical co-ordinates). The algorithm is based on the Sag–Szekeres method (see Sag and Szekeres 1964), applying the product trapezoidal formula after a suitable radial transformation. An improved transformation technique is developed: depending on the behaviour of the function and on the required accuracy, different transformations can be used, some of which are 'double exponential', as defined by Takahasi and Mori 1974. The resulting technique allows the function to deal with integrand singularities on the surface or at the centre of the sphere. When the estimated error of the approximation with mesh size h is larger than the tolerated error, the trapezoidal formula with mesh size $h/2$ is calculated. A drawback of this method is the exponential growth of the number of function evaluations in the successive approximations (this number grows with a factor $\approx 2^n$). This introduces the restriction $n \leq 4$. Because the convergence rate of the successive approximations is normally better than linear, the error estimate is based on the linear extrapolation of the difference between the successive approximations (see Robinson and de Doncker 1981 and Roose and de Doncker 1981). For further details of the algorithm, see Roose and de Doncker 1981.

4 References

Robinson I and de Doncker E 1981 Automatic computation of improper integrals over a bounded or unbounded planar region *Computing* **27** 89–284

Roose D and de Doncker E 1981 Automatic integration over a sphere *J. Comput. Appl. Math.* **7** 203–224

Sag T W and Szekeres G 1964 Numerical evaluation of high-dimensional integrals *Math. Comput.* **18** 245–253

Takahasi H and Mori M 1974 *Double Exponential Formulas for Numerical Integration* **9** Publ. RIMS, Kyoto University 721–741

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(ndim, x)
```

Input Parameters

1: **ndim – int32 scalar**

n, the number of dimensions of the integral.

2: **x(ndim) – double array**

The co-ordinates of the point at which the integrand f must be evaluated. These co-ordinates are given in Cartesian or spherical polar form according to the value of **icoord**.

Output Parameters

1: **result – double scalar**

The result of the function.

See also Section 8.

2: **ndim – int32 scalar**

n, the dimension of the sphere.

Constraint: $2 \leq$ **ndim** ≤ 4.

3: **radius – double scalar**

α, the radius of the sphere.

Constraint: **radius** ≥ 0.0.

4: **epsa – double scalar**

The requested absolute tolerance. If **epsa** < 0.0, its absolute value is used. See Section 7.

5: **epsr – double scalar**

The requested relative tolerance.

epsr < 0.0

Its absolute value is used.

epsr $< 10 \times$ (***machine precision***)

The latter value is used as **epsr** by the function. See Section 7.

6: **icoord – int32 scalar**

Must specify which kind of co-ordinates are used in user-supplied real function **f**.

icoord $= 0$

Cartesian co-ordinates x_i, for $i = 1, 2, \ldots, n$.

icoord $= 1$

Spherical co-ordinates (see Section 8.2): $\mathbf{x}(1) = \rho$; $\mathbf{x}(i) = \theta_{i-1}$, for $i = 2, 3, \ldots, n$.

icoord $= 2$,

Special spherical polar co-ordinates (see Section 8.3), with the additional transformation $\rho = \alpha - \lambda$: $\mathbf{x}(1) = \lambda = \alpha - \rho$; $\mathbf{x}(i) = \theta_{i-1}$, for $i = 2, 3, \ldots, n$.

Constraint: **icoord** $= 0$, 1 or 2.

If **method** $= 3$ or 4, **icoord** $= 2$

5.2 Optional Input Parameters

1: **method – int32 scalar**

Must specify the transformation to be used by the function. The choice depends on the behaviour of the integrand and on the required accuracy.

For well-behaved functions and functions with mild singularities on the surface of the sphere only:

method $= 1$

Low accuracy required.

method $= 2$

High accuracy required.

For functions with severe singularities on the surface of the sphere only:

method $= 3$

Low accuracy required.

method $= 4$

High accuracy required.

(in this case **icoord** must be set to 2, and the function defined in special spherical co-ordinates).

(When **method** $= 4$, **icoord** must be set to 2, and the function defined in special spherical co-ordinates).

For functions with a singularity at the centre of the sphere (and possibly with singularities on the surface as well):

method $= 5$

Low accuracy required.

method $= 6$

High accuracy required.

method $= 0$ can be used as a default value and is equivalent to **method** $= 1$ if **epsr** $> 10^{-6}$, and to **method** $= 2$ if **epsr** $\leq 10^{-6}$.

The distinction between low and high required accuracies, as mentioned above, depends also on the behaviour of the function. Roughly one may assume the critical value of **epsa** and **epsr** to be 10^{-6}, but the critical value will be smaller for a well-behaved integrand and larger for an integrand with severe singularities.

Suggested value: **method** $= 0$.

Default: 0

Constraint: $0 \leq$ **method** ≤ 6. If **icoord** $= 2$, **method** $= 3$ or 4.

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: **esterr – double scalar**

An estimate of the modulus of the absolute error.

3: **nevals – int32 scalar**

The number of function evaluations used.

4: **ifail – int32 scalar**

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

6 Error Indicators and Warnings

Note: d01ja may return useful information for one or more of the following detected errors or warnings.

ifail $= 1$

The required accuracy cannot be achieved within a limiting number of function evaluations (which is set by the function).

ifail $= 2$

The required accuracy cannot be achieved because of round-off error.

ifail $= 3$

The required accuracy cannot be achieved because the maximum accuracy with respect to the machine constants x02aj and x02am has been attained. If this maximum accuracy is rather low (compared with x02aj), the cause of the problem is a severe singularity on the boundary or at the centre of the sphere. If **method** $= 0$, 1 or 2, then setting **method** $= 3$ or 4 may help.

ifail $= 4$

On entry, **ndim** < 2 or **ndim** > 4,
or **radius** < 0.0,
or **method** < 0 or **method** > 6,
or **icoord** < 0 or **icoord** > 2,
or **icoord** $= 2$ and **method** $\neq 3$ or 4,
or **method** $= 3$ or 4 and **icoord** $\neq 2$.

No calculations have been performed. **result** and **esterr** are set to 0.0.

7 Accuracy

You can specify an absolute and/or a relative tolerance, setting **epsa** and **epsr**. The function attempts to calculate an approximation **result** such that

$$|I - \mathbf{result}| \leq \max\{\mathbf{epsa}, \mathbf{epsr} \times |I|\}.$$

If $0 \leq \mathbf{ifail} \leq 3$, **esterr** returns an estimate of, but not necessarily a bound for, $|I - \mathbf{result}|$.

8 Further Comments

8.1 Timing

Timing depends on the integrand and the accuracy required.

8.2 Spherical Polar Co-ordinates

Cartesian co-ordinates are related to the spherical polar co-ordinates by:

$$\begin{aligned} x_1 &= \rho.\sin\theta_1 \cdots \sin\theta_{n-2}.\sin\theta_{n-1} \\ x_2 &= \rho.\sin\theta_1 \cdots \sin\theta_{n-2}.\cos\theta_{n-1} \\ x_3 &= \rho.\sin\theta_1 \cdots \cos\theta_{n-2} \\ &\vdots \\ x_n &= \rho.\cos\theta_1 \end{aligned}$$

where $0 < \theta_i < \pi$, for $i = 1, 2, \ldots, n-2$ and $0 < \theta_{n-1} < 2\pi$.

8.3 Machine Dependencies

As a consequence of the transformation technique, the severity of the singularities which can be handled by d01ja depends on the precision and range of real numbers on the machine. **method** $= 3$ or 4 must be used when the singularity on the surface is 'severe' in view of the requested accuracy and ***machine precision***. In practice one has to set **method** $= 3$ or 4 if d01ja terminates with **ifail** $= 3$ when called with **method** $= 0$, 1 or 2.

When integrating a function with a severe singular behaviour on the surface of the sphere, the additional transformation $\rho = \alpha - \lambda$ helps to avoid the loss of significant figures due to round-off error in the calculation of the integration nodes which are very close to the surface. For these points, the value of λ can be computed more accurately than the value of ρ. Naturally, care must be taken that the function subprogram does not contain expressions of the form $\alpha - \lambda$, which could cause a large round-off error in the calculation of the integrand at the boundary of the sphere.

Care should be taken to avoid underflow and/or overflow problems in the function subprogram, because some of the integration nodes used by d01ja may be very close to the surface or to the centre of the sphere.

Example:

suppose the function

$$f(\rho) = \left(1 - \rho^2\right)^{-0.7}$$

is to be integrated over the unit sphere, with **method** $= 3$ or 4. Then **icoord** should be set to 2; the transformation $\rho = 1 - \lambda$ gives $f(\rho) = \left(2\lambda - \lambda^2\right)^{-0.7}$; and user-supplied real function **f** could be coded thus:

```
function result = f(ndim, x)
  result = 1;
  a = x(1);
  if (a > 0)
    result = 1/(a*(2-a))^0.7;
  end
```

Note that d01ja ensures that $\lambda = \mathbf{x}(1) >$ x02am, but underflow could occur in the computation of λ^2.

9 Example

d01ja_f.m

```
function result = f(ndim, x)
  a = (1-x(1))*(1+x(1));
  if (a == 0)
    result = 0;
  else
    result = 1/sqrt(a);
  end
```

```
ndim = int32(2);
radius = 1;
epsa = 0;
epsr = 5e-05;
icoord = int32(1);
[result, esterr, nevals, ifail] = ...
    d01ja('d01ja_f', ndim, radius, epsa, epsr, icoord)
```

```
result =
    6.2832
esterr =
   1.9768e-04
nevals =
         193
ifail =
           0
```