NAG Toolbox for MATLAB

e01da

1 Purpose

e01da computes a bicubic spline interpolating surface through a set of data values, given on a rectangular grid in the x-y plane.

2 Syntax

```
[px, py, lamda, mu, c, ifail] = e01da(x, y, f, 'mx', mx, 'my', my)
```

3 Description

e01da determines a bicubic spline interpolant to the set of data points $\left(x_q, y_r, f_{q,r}\right)$, for $q = 1, 2, \ldots, m_x$ and $r = 1, 2, \ldots, m_y$. The spline is given in the B-spline representation

$$s(x,y) = \sum_{i=1}^{m_x} \sum_{j=1}^{m_y} c_{ij} M_i(x) N_j(y),$$

such that

$$s\left(x_q, y_r\right) = f_{q,r},$$

where $M_i(x)$ and $N_j(y)$ denote normalized cubic B-splines, the former defined on the knots λ_i to λ_{i+4} and the latter on the knots μ_j to μ_{j+4}, and the c_{ij} are the spline coefficients. These knots, as well as the coefficients, are determined by the function, which is derived from the function B2IRE in Anthony *et al.* 1982. The method used is described in Section 8.2.

For further information on splines, see Hayes and Halliday 1974 for bicubic splines and de Boor 1972 for normalized B-splines.

Values of the computed spline can subsequently be obtained by calling e02de or e02df as described in Section 8.3.

4 References

Anthony G T, Cox M G and Hayes J G 1982 *DASL – Data Approximation Subroutine Library* National Physical Laboratory

Cox M G 1975a An algorithm for spline interpolation *J. Inst. Math. Appl.* **15** 95–108

de Boor C 1972 On calculating with B-splines *J. Approx. Theory* **6** 50–62

Hayes J G and Halliday J 1974 The least-squares fitting of cubic spline surfaces to general data sets *J. Inst. Math. Appl.* **14** 89–103

5 Parameters

5.1 Compulsory Input Parameters

1: **x**(**mx**) – **double array**

2: **y**(**my**) – **double array**

x(q) and **y**(r) must contain x_q, for $q = 1, 2, \ldots, m_x$, and y_r, for $r = 1, 2, \ldots, m_y$, respectively.

Constraints:

$\mathbf{x}(q) < \mathbf{x}(q+1)$, for $q = 1, 2, \ldots, m_x - 1$;
$\mathbf{y}(r) < \mathbf{y}(r+1)$, for $r = 1, 2, \ldots, m_y - 1$.

3: **f(mx × my) – double array**

$\mathbf{f}\left(m_y \times (q-1) + r\right)$ must contain $f_{q,r}$, for $q = 1, 2, \ldots, m_x$ and $r = 1, 2, \ldots, m_y$.

5.2 Optional Input Parameters

1: **mx – int32 scalar**

2: **my – int32 scalar**

Default: The dimension of the arrays **x**, **y**. (An error is raised if these dimensions are not equal.)

mx and **my** must specify m_x and m_y respectively, the number of points along the x and y axis that define the rectangular grid.

Constraint: $\mathbf{mx} \geq 4$ and $\mathbf{my} \geq 4$.

5.3 Input Parameters Omitted from the MATLAB Interface

wrk

5.4 Output Parameters

1: **px – int32 scalar**

2: **py – int32 scalar**

px and **py** contain $m_x + 4$ and $m_y + 4$, the total number of knots of the computed spline with respect to the x and y variables, respectively.

3: **lamda(mx + 4) – double array**

4: **mu(my + 4) – double array**

lamda contains the complete set of knots λ_i associated with the x variable, i.e., the interior knots $\mathbf{lamda}(5), \mathbf{lamda}(6), \ldots, \mathbf{lamda}(\mathbf{px} - 4)$, as well as the additional knots

$$\mathbf{lamda}(1) = \mathbf{lamda}(2) = \mathbf{lamda}(3) = \mathbf{lamda}(4) = \mathbf{x}(1)$$

and

$$\mathbf{lamda}(\mathbf{px} - 3) = \mathbf{lamda}(\mathbf{px} - 2) = \mathbf{lamda}(\mathbf{px} - 1) = \mathbf{lamda}(\mathbf{px}) = \mathbf{x}(\mathbf{mx})$$

needed for the B-spline representation. **mu** contains the corresponding complete set of knots μ_i associated with the y variable.

5: **c(mx × my) – double array**

The coefficients of the spline interpolant. $\mathbf{c}\left(m_y \times (i-1) + j\right)$ contains the coefficient c_{ij} described in Section 3.

6: **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 $= 1$

On entry, **mx** < 4,
or **my** < 4.

ifail $= 2$

On entry, either the values in the **x** array or the values in the **y** array are not in increasing order if not already there.

ifail $= 3$

A system of linear equations defining the B-spline coefficients was singular; the problem is too ill-conditioned to permit solution.

7 Accuracy

The main sources of rounding errors are in steps 2, 3, 6 and 7 of the algorithm described in Section 8.2. It can be shown (see Cox 1975a) that the matrix A_x formed in step 2 has elements differing relatively from their true values by at most a small multiple of 3ϵ, where ϵ is the ***machine precision***. A_x is 'totally positive', and a linear system with such a coefficient matrix can be solved quite safely by elimination without pivoting. Similar comments apply to steps 6 and 7. Thus the complete process is numerically stable.

8 Further Comments

8.1 Timing

The time taken by e01da is approximately proportional to $m_x m_y$.

8.2 Outline of method used

The process of computing the spline consists of the following steps:

1. choice of the interior x-knots $\lambda_5, \lambda_6, \ldots, \lambda_{m_x}$ as $\lambda_i = x_{i-2}$, for $i = 5, 6, \ldots, m_x$,
2. formation of the system
$$A_x E = F,$$
where A_x is a band matrix of order m_x and bandwidth 4, containing in its qth row the values at x_q of the B-splines in x, $\mathbf{f}$ is the m_x by m_y rectangular matrix of values $f_{q,r}$, and E denotes an m_x by m_y rectangular matrix of intermediate coefficients,
3. use of Gaussian elimination to reduce this system to band triangular form,
4. solution of this triangular system for E,
5. choice of the interior y knots $\mu_5, \mu_6, \ldots, \mu_{m_y}$ as $\mu_i = y_{i-2}$, for $i = 5, 6, \ldots, m_y$,
6. formation of the system
$$A_y C^{\mathrm{T}} = E^{\mathrm{T}},$$
where A_y is the counterpart of A_x for the y variable, and C denotes the m_x by m_y rectangular matrix of values of c_{ij},
7. use of Gaussian elimination to reduce this system to band triangular form,
8. solution of this triangular system for C^{T} and hence C.

For computational convenience, steps 2 and 3, and likewise steps 6 and 7, are combined so that the formation of A_x and A_y and the reductions to triangular form are carried out one row at a time.

8.3 Evaluation of Computed Spline

The values of the computed spline at the points $(\mathbf{x}(r), \mathbf{y}(r))$, for $r = 1, 2, \ldots, \mathbf{m}$N, may be obtained in the double array **ff**, of length at least **m**, by the following call:

```
[ff, ifail] = e02de(x, y, lamda, mu, c);
```

where **lamda**, **mu** and **c** are the output parameters of e01da (see e02de).

To evaluate the computed spline on an **mx** by **my** rectangular grid of points in the *x*-*y* plane, which is defined by the *x* co-ordinates stored in $\mathbf{x}(q)$, for $q = 1, 2, \ldots, \mathbf{mx}$, and the *y* co-ordinates stored in $\mathbf{y}(r)$, for $r = 1, 2, \ldots, \mathbf{my}$, returning the results in the double array **ff** which is of length at least $\mathbf{mx} \times \mathbf{my}$, the following call may be used:

```
[fg, ifail] = e02df(x, y, lamda, mu, c);
```

where **lamda**, **mu** and **c** are the output parameters of e01da. The result of the spline evaluated at grid point (q, r) is returned in element $(\mathbf{my} \times (q - 1) + r)$ of the array **ff**. (See e02df.)

9 Example

```
x = [1;
     1.1;
     1.3;
     1.5;
     1.6;
     1.8;
     2];
y = [0;
     0.1;
     0.4;
     0.7;
     0.9;
     1];
f = [1;
     1.1;
     1.4;
     1.7;
     1.9;
     2;
     1.21;
     1.31;
     1.61;
     1.91;
     2.11;
     2.21;
     1.69;
     1.79;
     2.09;
     2.39;
     2.59;
     2.69;
     2.25;
     2.35;
     2.65;
     2.95;
     3.15;
     3.25;
     2.56;
     2.66;
     2.96;
     3.26;
     3.46;
```

```
     3.56;
     3.24;
     3.34;
     3.64;
     3.94;
     4.14;
     4.24;
     4;
     4.1;
     4.4;
     4.7;
     4.9;
     5];
[px, py, lamda, mu, c, ifail] = e01da(x, y, f)
```

```
px =
          11
py =
          10
lamda =
    1.0000
    1.0000
    1.0000
    1.0000
    1.3000
    1.5000
    1.6000
    2.0000
    2.0000
    2.0000
    2.0000
mu =
         0
         0
         0
         0
    0.4000
    0.7000
    1.0000
    1.0000
    1.0000
    1.0000
c =
    1.0000
    1.1333
    1.3667
    1.7000
    1.9000
    2.0000
    1.2000
    1.3333
    1.5667
    1.9000
    2.1000
    2.2000
    1.5833
    1.7167
    1.9500
    2.2833
    2.4833
    2.5833
    2.1433
    2.2767
    2.5100
    2.8433
    3.0433
    3.1433
    2.8667
    3.0000
    3.2333
```

```
    3.5667
    3.7667
    3.8667
    3.4667
    3.6000
    3.8333
    4.1667
    4.3667
    4.4667
    4.0000
    4.1333
    4.3667
    4.7000
    4.9000
    5.0000
ifail =
           0
```