NAG Toolbox for MATLAB

e02dd

1 Purpose

e02dd computes a bicubic spline approximation to a set of scattered data. The knots of the spline are located automatically, but a single parameter must be specified to control the trade-off between closeness of fit and smoothness of fit.

2 Syntax

```
[nx, lamda, ny, mu, c, fp, rank, wrk, ifail] = e02dd(start, x, y, f, w,
s, nx, lamda, ny, mu, wrk, 'm', m, 'nxest', nxest, 'nyest', nyest,
'lwrk', lwrk)
```

3 Description

e02dd determines a smooth bicubic spline approximation $s(x,y)$ to the set of data points (x_r,y_r,f_r) with weights w_r, for $r = 1,2,\ldots,m$.

The approximation domain is considered to be the rectangle $[x_{\min},x_{\max}] \times [y_{\min},y_{\max}]$, where $x_{\min}$ ($y_{\min}$) and $x_{\max}$ ($y_{\max}$) denote the lowest and highest data values of $x(y)$.

The spline is given in the B-spline representation

$$s(x,y) = \sum_{i=1}^{n_x-4}\sum_{j=1}^{n_y-4} c_{ij}M_i(x)N_j(y), \tag{1}$$

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

The total numbers n_x and n_y of these knots and their values $\lambda_1,\ldots,\lambda_{n_x}$ and $\mu_1,\ldots,\mu_{n_y}$ are chosen automatically by the function. The knots $\lambda_5,\ldots,\lambda_{n_x-4}$ and $\mu_5,\ldots,\mu_{n_y-4}$ are the interior knots; they divide the approximation domain $[x_{\min},x_{\max}] \times [y_{\min},y_{\max}]$ into $(n_x-7)\times(n_y-7)$ subpanels $[\lambda_i,\lambda_{i+1}] \times \left[\mu_j,\mu_{j+1}\right]$, for $i = 4,5,\ldots,n_x-4$; $j = 4,5,\ldots,n_y-4$. Then, much as in the curve case (see e02be), the coefficients c_{ij} are determined as the solution of the following constrained minimization problem:

minimize

$$\eta, \tag{2}$$

subject to the constraint

$$\theta = \sum_{r=1}^{m}\epsilon_r^2 \leq S \tag{3}$$

where:

η
is a measure of the (lack of) smoothness of $s(x,y)$. Its value depends on the discontinuity jumps in $s(x,y)$ across the boundaries of the subpanels. It is zero only when there are no discontinuities and is positive otherwise, increasing with the size of the jumps (see Dierckx 1981b for details).

ϵ_r
denotes the weighted residual $w_r(f_r - s(x_r,y_r))$,

and

s is a nonnegative number to be specified by you.

By means of the parameter **s**, 'the smoothing factor', you will then control the balance between smoothness and closeness of fit, as measured by the sum of squares of residuals in (3). If **s** is too large, the spline will be too smooth and signal will be lost (underfit); if **s** is too small, the spline will pick up too much noise (overfit). In the extreme cases the method would return an interpolating spline $(\theta = 0)$ if **s** were set to zero, and returns the least-squares bicubic polynomial $(\eta = 0)$ if **s** is set very large. Experimenting with **s**-values between these two extremes should result in a good compromise. (See Section 8.2 for advice on choice of **s**.) Note however, that this function, unlike e02be and e02dc, does not allow **s** to be set exactly to zero: to compute an interpolant to scattered data, e01sa or e01sg should be used.

The method employed is outlined in Section 8.5 and fully described in Dierckx 1981b and Dierckx 1981a. It involves an adaptive strategy for locating the knots of the bicubic spline (depending on the function underlying the data and on the value of **s**), and an iterative method for solving the constrained minimization problem once the knots have been determined.

Values of the computed spline can subsequently be computed by calling e02de or e02df as described in Section 8.6.

4 References

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

Dierckx P 1981a An improved algorithm for curve fitting with spline functions *Report TW54* Department of Computer Science, Katholieke Univerciteit Leuven

Dierckx P 1981b An algorithm for surface fitting with spline functions *IMA J. Numer. Anal.* **1** 267–283

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

Peters G and Wilkinson J H 1970 The least-squares problem and pseudo-inverses *Comput. J.* **13** 309–316

Reinsch C H 1967 Smoothing by spline functions *Numer. Math.* **10** 177–183

5 Parameters

5.1 Compulsory Input Parameters

1: **start – string**

Must be set to 'C' or 'W'.

start = 'C' (Cold start)

The function will build up the knot set starting with no interior knots. No values need be assigned to the parameters **nx**, **ny**, **lamda**, **mu** or **wrk**.

start = 'W' (Warm start)

The function will restart the knot-placing strategy using the knots found in a previous call of the function. In this case, the parameters **nx**, **ny**, **lamda**, **mu** and **wrk** must be unchanged from that previous call. This warm start can save much time in searching for a satisfactory value of **s**.

Constraint: **start** = 'C' or 'W'.

2: **x(m) – double array**

3: **y(m) – double array**

4: **f(m) – double array**

x(r), **y**(r), **f**(r) must be set to the co-ordinates of (x_r, y_r, f_r), the rth data point, for $r = 1, 2, \ldots, m$. The order of the data points is immaterial.

5: **w(m) – double array**

w(r) must be set to w_r, the rth value in the set of weights, for $r = 1, 2, \ldots, m$. Zero weights are permitted and the corresponding points are ignored, except when determining $x_{\min}$, $x_{\max}$, $y_{\min}$ and $y_{\max}$ (see Section 8.4). For advice on the choice of weights, see Section 2.1.2 in the E02 Chapter Introduction.

Constraint: the number of data points with nonzero weight must be at least 16.

6: **s – double scalar**

The smoothing factor, **s**.

For advice on the choice of **s**, see Sections 3 and 8.2.

Constraint: **s** ≥ 0.0.

7: **nx – int32 scalar**

If the warm start option is used, the value of **nx** must be left unchanged from the previous call.

8: **lamda(nxest) – double array**

If the warm start option is used, the values **lamda**$(1), \ldots,$ **lamda**$($**nx**$)$ must be left unchanged from the previous call.

9: **ny – int32 scalar**

If the warm start option is used, the value of **ny** must be left unchanged from the previous call.

10: **mu(nyest) – double array**

If the warm start option is used, the values **mu**$(1), \ldots,$ **mu**$($**ny**$)$ must be left unchanged from the previous call.

11: **wrk(lwrk) – double array**

If the warm start option is used, on entry, the value of **wrk**(1) must be left unchanged from the previous call.

This array is used as workspace.

5.2 Optional Input Parameters

1: **m – int32 scalar**

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

m, the number of data points.

The number of data points with nonzero weight (see **w**) must be at least 16.

2: **nxest – int32 scalar**

3: **nyest – int32 scalar**

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

an upper bound for the number of knots n_x and n_y required in the x- and y-directions respectively.

In most practical situations, **nxest** = **nyest** $= 4 + \sqrt{m/2}$ is sufficient. See also Section 8.3.

Constraint: **nxest** ≥ 8 and **nyest** ≥ 8.

4: **lwrk – int32 scalar**

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

Constraint: **lwrk** $\geq (7 \times u \times v + 25 \times w) \times (w + 1) + 2 \times (u + v + 4 \times \mathbf{m}) + 23 \times w + 56$, where $v = \mathbf{nyest} - 4$, $u = \mathbf{nxest} - 4$, and $w = \max(u, v)$,

where $u = \mathbf{nxest} - 4$, where $u = \mathbf{nxest} - 4$, $v = \mathbf{nyest} - 4$, where $v = \mathbf{nyest} - 4$, and $w = \max(u, v)$, where $v = \mathbf{nyest} - 4$, $u = \mathbf{nxest} - 4$, and $w = \max(u, v)$.

For some problems, the function may need to compute the minimal least-squares solution of a rank-deficient system of linear equations (see Section 3). The amount of workspace required to solve such problems will be larger than specified by the value given above, which must be increased by an amount, LWRK2 say. An upper bound for LWRK2 is given by $4 \times u \times v \times w + 2 \times u \times v + 4 \times w$, where $v = \mathbf{nyest} - 4$, $u = \mathbf{nxest} - 4$, and $w = \max(u, v)$, where u, where $u = \mathbf{nxest} - 4$, v, where $v = \mathbf{nyest} - 4$ and w, where $w = \max(u, v)$ are as above. However, if there are enough data points, scattered uniformly over the approximation domain, and if the smoothing factor **s** is not too small, there is a good chance that this extra workspace is not needed. A lot of memory might therefore be saved by assuming $LWRK2 = 0$

5.3 Input Parameters Omitted from the MATLAB Interface

iwrk, liwrk

5.4 Output Parameters

1: **nx – int32 scalar**

The total number of knots, n_x, of the computed spline with respect to the x variable.

2: **lamda**(**nxest**) **– double array**

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

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

and

$$\mathbf{lamda}(\mathbf{nx} - 3) = \mathbf{lamda}(\mathbf{nx} - 2) = \mathbf{lamda}(\mathbf{nx} - 1) = \mathbf{lamda}(\mathbf{nx}) = x_{\max}$$

needed for the B-spline representation (where $x_{\min}$ and $x_{\max}$ are as described in Section 3).

3: **ny – int32 scalar**

The total number of knots, n_y, of the computed spline with respect to the y variable.

4: **mu**(**nyest**) **– double array**

Contains the complete set of knots μ_i associated with the y variable, i.e., the interior knots $\mathbf{mu}(5), \mathbf{mu}(6), \ldots, \mathbf{mu}(\mathbf{ny} - 4)$ as well as the additional knots

$$\mathbf{mu}(1) = \mathbf{mu}(2) = \mathbf{mu}(3) = \mathbf{mu}(4) = y_{\min}$$

and

$$\mathbf{mu}(\mathbf{ny} - 3) = \mathbf{mu}(\mathbf{ny} - 2) = \mathbf{mu}(\mathbf{ny} - 1) = \mathbf{mu}(\mathbf{ny}) = y_{\max}$$

needed for the B-spline representation (where $y_{\min}$ and $y_{\max}$ are as described in Section 3).

5: **c**((**nxest** − **4**) × (**nyest** − **4**)) **– double array**

The coefficients of the spline approximation. $\mathbf{c}\big(\big(n_y - 4\big) \times (i - 1) + j\big)$ is the coefficient c_{ij} defined in Section 3.

6: **fp – double scalar**

The weighted sum of squared residuals, θ, of the computed spline approximation. **fp** should equal **s** within a relative tolerance of 0.001 unless $\mathbf{nx} = \mathbf{ny} = 8$, when the spline has no interior knots and so is simply a bicubic polynomial. For knots to be inserted, **s** must be set to a value below the value of **fp** produced in this case.

7: **rank – int32 scalar**

Gives the rank of the system of equations used to compute the final spline (as determined by a suitable machine-dependent threshold). When $\mathbf{rank} = (\mathbf{nx} - 4) \times (\mathbf{ny} - 4)$, the solution is unique; otherwise the system is rank-deficient and the minimum-norm solution is computed. The latter case may be caused by too small a value of **s**.

8: **wrk(lwrk) – double array**

If the warm start option is used, on entry, the value of $\mathbf{wrk}(1)$ must be left unchanged from the previous call.

This array is used as workspace.

9: **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, **start** $\neq$ 'C' or 'W',
or the number of data points with nonzero weight < 16,
or $\mathbf{s} \leq 0.0$,
or $\mathbf{nxest} < 8$,
or $\mathbf{nyest} < 8$,
or $\mathbf{lwrk} < (7 \times u \times v + 25 \times w) \times (w + 1) + 2 \times (u + v + 4 \times \mathbf{m}) + 23 \times w + 56$, where $u = \mathbf{nxest} - 4$, $v = \mathbf{nyest} - 4$ and $w = \max(u, v)$,
or $\mathbf{liwrk} < \mathbf{m} + 2 \times (\mathbf{nxest} - 7) \times (\mathbf{nyest} - 7)$.

ifail $= 2$

On entry, either all the $\mathbf{x}(r)$, for $r = 1, 2, \ldots, \mathbf{m}$, are equal, or all the $\mathbf{y}(r)$, for $r = 1, 2, \ldots, \mathbf{m}$, are equal.

ifail $= 3$

The number of knots required is greater than allowed by **nxest** and **nyest**. Try increasing **nxest** and/or **nyest** and, if necessary, supplying larger arrays for the parameters **lamda**, **mu**, **c**, **wrk** and **iwrk**. However, if **nxest** and **nyest** are already large, say **nxest**, **nyest** $> 4 + \sqrt{\mathbf{m}/2}$, then this error exit may indicate that **s** is too small.

ifail $= 4$

No more knots can be added because the number of B-spline coefficients $(\mathbf{nx} - 4) \times (\mathbf{ny} - 4)$ already exceeds the number of data points **m**. This error exit may occur if either of **s** or **m** is too small.

ifail $= 5$

No more knots can be added because the additional knot would (quasi) coincide with an old one. This error exit may occur if too large a weight has been given to an inaccurate data point, or if **s** is too small.

ifail $= 6$

The iterative process used to compute the coefficients of the approximating spline has failed to converge. This error exit may occur if **s** has been set very small. If the error persists with increased **s**, consult NAG.

ifail $= 7$

lwrk is too small; the function needs to compute the minimal least-squares solution of a rank-deficient system of linear equations, but there is not enough workspace. There is no approximation returned but, having saved the information contained in **nx**, **lamda**, **ny**, **mu** and **wrk**, and having adjusted the value of **lwrk** and the dimension of array **wrk** accordingly, you can continue at the point the program was left by calling e02dd with **start** = 'W'. Note that the requested value for **lwrk** is only large enough for the current phase of the algorithm. If the function is restarted with **lwrk** set to the minimum value requested, a larger request may be made at a later stage of the computation. See Section 5 for the upper bound on **lwrk**. On soft failure, the minimum requested value for **lwrk** is returned in **iwrk**(1) and the safe value for **lwrk** is returned in **iwrk**(2).

If **ifail** $= 3$, 4, 5 or 6, a spline approximation is returned, but it fails to satisfy the fitting criterion (see (2) and (3) in Section 3 – perhaps only by a small amount, however.

7 Accuracy

On successful exit, the approximation returned is such that its weighted sum of squared residuals **fp** is equal to the smoothing factor **s**, up to a specified relative tolerance of 0.001 – except that if $n_x = 8$ and $n_y = 8$, **fp** may be significantly less than **s**: in this case the computed spline is simply the least-squares bicubic polynomial approximation of degree 3, i.e., a spline with no interior knots.

8 Further Comments

8.1 Timing

The time taken for a call of e02dd depends on the complexity of the shape of the data, the value of the smoothing factor **s**, and the number of data points. If e02dd is to be called for different values of **s**, much time can be saved by setting **start** = 'W' after the first call.

It should be noted that choosing **s** very small considerably increases computation time.

8.2 Choice of s

If the weights have been correctly chosen (see Section 2.1.2 in the E02 Chapter Introduction), the standard deviation of $w_r f_r$ would be the same for all r, equal to σ, say. In this case, choosing the smoothing factor **s** in the range $\sigma^2\left(m \pm \sqrt{2m}\right)$, as suggested by Reinsch 1967, is likely to give a good start in the search for a satisfactory value. Otherwise, experimenting with different values of **s** will be required from the start.

In that case, in view of computation time and memory requirements, it is recommended to start with a very large value for **s** and so determine the least-squares bicubic polynomial; the value returned for **fp**, call it $\mathbf{fp}_0$, gives an upper bound for **s**. Then progressively decrease the value of **s** to obtain closer fits – say by a factor of 10 in the beginning, i.e., $\mathbf{s} = \mathbf{fp}_0/10$, $\mathbf{s} = \mathbf{fp}_0/100$, and so on, and more carefully as the approximation shows more details.

To choose **s** very small is strongly discouraged. This considerably increases computation time and memory requirements. It may also cause rank-deficiency (as indicated by the parameter **rank**) and endanger numerical stability.

The number of knots of the spline returned, and their location, generally depend on the value of **s** and on the behaviour of the function underlying the data. However, if e02dd is called with **start** = 'W', the knots returned may also depend on the smoothing factors of the previous calls. Therefore if, after a number of trials with different values of **s** and **start** = 'W', a fit can finally be accepted as satisfactory, it may be worthwhile to call e02dd once more with the selected value for **s** but now using **start** = 'C'. Often, e02dd

then returns an approximation with the same quality of fit but with fewer knots, which is therefore better if data reduction is also important.

8.3 Choice of nxest and nyest

The number of knots may also depend on the upper bounds **nxest** and **nyest**. Indeed, if at a certain stage in e02dd the number of knots in one direction (say n_x) has reached the value of its upper bound (**nxest**), then from that moment on all subsequent knots are added in the other (y) direction. This may indicate that the value of **nxest** is too small. On the other hand, it gives you the option of limiting the number of knots the function locates in any direction. For example, by setting **nxest** $= 8$ (the lowest allowable value for **nxest**), you can indicate that he wants an approximation which is a simple cubic polynomial in the variable x.

8.4 Restriction of the approximation domain

The fit obtained is not defined outside the rectangle $\left[\lambda_4, \lambda_{n_x-3}\right] \times \left[\mu_4, \mu_{n_y-3}\right]$. The reason for taking the extreme data values of x and y for these four knots is that, as is usual in data fitting, the fit cannot be expected to give satisfactory values outside the data region. If, nevertheless, you require values over a larger rectangle, this can be achieved by augmenting the data with two artificial data points $(a, c, 0)$ and $(b, d, 0)$ with zero weight, where $[a, b] \times [c, d]$ denotes the enlarged rectangle.

8.5 Outline of method used

First suitable knot sets are built up in stages (starting with no interior knots in the case of a cold start but with the knot set found in a previous call if a warm start is chosen). At each stage, a bicubic spline is fitted to the data by least-squares and θ, the sum of squares of residuals, is computed. If $\theta > \mathbf{s}$, a new knot is added to one knot set or the other so as to reduce θ at the next stage. The new knot is located in an interval where the fit is particularly poor. Sooner or later, we find that $\theta \leq \mathbf{s}$ and at that point the knot sets are accepted. The function then goes on to compute a spline which has these knot sets and which satisfies the full fitting criterion specified by (2) and (3). The theoretical solution has $\theta = \mathbf{s}$. The function computes the spline by an iterative scheme which is ended when $\theta = \mathbf{s}$ within a relative tolerance of 0.001. The main part of each iteration consists of a linear least-squares computation of special form, done in a similarly stable and efficient manner as in e02da. As there also, the minimal least-squares solution is computed wherever the linear system is found to be rank-deficient.

An exception occurs when the function finds at the start that, even with no interior knots (N $= 8$), the least-squares spline already has its sum of squares of residuals $\leq \mathbf{s}$. In this case, since this spline (which is simply a bicubic polynomial) also has an optimal value for the smoothness measure η, namely zero, it is returned at once as the (trivial) solution. It will usually mean that **s** has been chosen too large.

For further details of the algorithm and its use see Dierckx 1981b.

8.6 Evaluation of computed spline

The values of the computed spline at the points $(\mathrm{TX}(r), \mathrm{TY}(r))$, for $r = 1, 2, \ldots, \mathrm{N}$, may be obtained in the double array **ff** (see e02de), of length at least N, by the following code:

```
[ff, ifail] = e02de(tx, ty, lamda, mu, c);
```

where **lamda**, **mu** and **c** are the output parameters of e02dd .

To evaluate the computed spline on a KX by KY rectangular grid of points in the x-y plane, which is defined by the x co-ordinates stored in $\mathrm{TX}(q)$, for $q = 1, 2, \ldots, \mathrm{KX}$, and the y co-ordinates stored in $\mathrm{TY}(r)$, for $r = 1, 2, \ldots, \mathrm{KY}$, returning the results in the double array FG (see e02df), which is of length at least KX $\times$ KY, the following call may be used:

```
[fg, ifail] = e02de(tx, ty, lamda, mu, c);
```

where **lamda**, **mu** and **c** are the output parameters of e02dd. The result of the spline evaluated at grid point (q, r) is returned in element $(\mathrm{KY} \times (q-1) + r)$ of the array FG.

9 Example

```
start = 'C';
x = [11.16;
     12.85;
     19.85;
     19.72;
     15.91;
     0;
     20.87;
     3.45;
     14.26;
     17.43;
     22.8;
     7.58;
     25;
     0;
     9.66;
     5.22;
     17.25;
     25;
     12.13;
     22.23;
     11.52;
     15.2;
     7.54;
     17.32;
     2.14;
     0.51;
     22.69;
     5.47;
     21.67;
     3.31];
y = [1.24;
     3.06;
     10.72;
     1.39;
     7.74;
     20;
     20;
     12.78;
     17.87;
     3.46;
     12.39;
     1.98;
     11.87;
     0;
     20;
     14.66;
     19.57;
     3.87;
     10.79;
     6.21;
     8.529999999999999;
     0;
     10.69;
     13.78;
     15.03;
     8.369999999999999;
     19.63;
     17.13;
     14.36;
     0.33];
f = [22.15;
     22.11;
     7.97;
     16.83;
     15.3;
```

```
     34.6;
     5.74;
     41.24;
     10.74;
     18.6;
     5.47;
     29.87;
     4.4;
     58.2;
     4.73;
     40.36;
     6.43;
     8.74;
     13.71;
     10.25;
     15.74;
     21.6;
     19.31;
     12.11;
     53.1;
     49.43;
     3.25;
     28.63;
     5.52;
     44.08];
w = [1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1;
     1];
s = 10;
nx = int32(0);
lamda = zeros(14,1);
ny = int32(0);
mu = zeros(14,1);
wrk = zeros(11016, 1);
[nxOut, lamdaOut, nyOut, muOut, c, fp, rank, wrkOut, ifail] = ...
    e02dd(start, x, y, f, w, s, nx, lamda, ny, mu, wrk)
```

```
nxOut =
          10
lamdaOut =
         0
         0
         0
```

```
         0
    9.7575
   18.2582
   25.0000
   25.0000
   25.0000
   25.0000
         0
         0
         0
         0
nyOut =
           9
muOut =
         0
         0
         0
         0
    9.0008
   20.0000
   20.0000
   20.0000
   20.0000
         0
         0
         0
         0
         0
c =
     array elided
fp =
   10.0021
rank =
          30
wrkOut =
     array elided
ifail =
           0
```