NAG Toolbox

nag_fit_2dspline_grid (e02dc)

1 Purpose

nag_fit_2dspline_grid (e02dc) computes a bicubic spline approximation to a set of data values, given on a rectangular grid in the x-y plane. The knots of the spline are located automatically, but a single argument must be specified to control the trade-off between closeness of fit and smoothness of fit.

2 Syntax

```
[nx, lamda, ny, mu, c, fp, wrk, iwrk, ifail] = nag_fit_2dspline_grid(start, x, y, f, s, nx, lamda, ny, mu, wrk, iwrk, 'mx', mx, 'my', my, 'nxest', nxest, 'nyest', nyest)
[nx, lamda, ny, mu, c, fp, wrk, iwrk, ifail] = e02dc(start, x, y, f, s, nx, lamda, ny, mu, wrk, iwrk, 'mx', mx, 'my', my, 'nxest', nxest, 'nyest', nyest)
```

Note: the interface to this routine has changed since earlier releases of the toolbox:

At Mark 22: *lwrk* and *liwrk* were removed from the interface.

3 Description

nag_fit_2dspline_grid (e02dc) determines a smooth bicubic spline approximation $s(x,y)$ 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

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

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 $[x_1,x_{m_x}]\times\left[y_1,y_{m_y}\right]$ into $(n_x-7)\times\left(n_y-7\right)$ subpanels $[\lambda_i,\lambda_{i+1}]\times\left[\mu_j,\mu_{j+1}\right]$, for $i = 4,5,\ldots,n_x-4$ and $j = 4,5,\ldots,n_y-4$. Then, much as in the curve case (see nag_fit_1dspline_auto (e02be)), the coefficients c_{ij} are determined as the solution of the following constrained minimization problem:

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

subject to the constraint

$$\theta = \sum_{q=1}^{m_x}\sum_{r=1}^{m_y}\epsilon_{q,r}^2 \leq S, \tag{3}$$

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 (1982) for details).

$\epsilon_{q,r}$ denotes the residual $f_{q,r} - s\left(x_q,y_r\right)$,

and S is a non-negative number specified by you.

By means of the argument 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 function will return an interpolating spline ($\theta = 0$) if S is set to zero, and 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 9.3 for advice on choice of S.)

The method employed is outlined in Section 9.5 and fully described in Dierckx (1981) and Dierckx (1982). 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 and derivatives of the computed spline can subsequently be computed by calling nag_fit_2dspline_evalv (e02de), nag_fit_2dspline_evalm (e02df) or nag_fit_2dspline_derivm (e02dh) as described in Section 9.6.

4 References

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

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

Dierckx P (1982) A fast algorithm for smoothing data on a rectangular grid while using spline functions *SIAM J. Numer. Anal.* **19** 1286–1304

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

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

5 Parameters

5.1 Compulsory Input Parameters

1: **start** – CHARACTER(1)

Determines whether calculations are to be performed afresh (Cold Start) or whether knots found in previous calls are to be used as an initial estimate of knot placement (Warm Start).

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

start = 'W'
The function will restart the knot-placing strategy using the knots found in a previous call of the function. In this case, the arguments **nx**, **ny**, **lamda**, **mu**, **wrk** and **iwrk** must be unchanged from that previous call. This warm start can save much time in determining a satisfactory set of knots for the given value of **s**. This is particularly useful when different smoothing factors are used for the same dataset.

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

2: **x**(**mx**) – REAL (KIND=nag_wp) array

x(q) must be set to x_q, the x coordinate of the qth grid point along the x axis, for $q = 1, 2, \ldots, m_x$.

Constraint: $x_1 < x_2 < \cdots < x_{m_x}$.

3: **y**(**my**) – REAL (KIND=nag_wp) array

y(r) must be set to y_r, the y coordinate of the rth grid point along the y axis, for $r = 1, 2, \ldots, m_y$.

Constraint: $y_1 < y_2 < \cdots < y_{m_y}$.

4: **f**(**mx** × **my**) – REAL (KIND=nag_wp) array

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

5: **s** – REAL (KIND=nag_wp)

The smoothing factor, S.

If **s** = 0.0, the function returns an interpolating spline.

If **s** is smaller than ***machine precision***, it is assumed equal to zero.

For advice on the choice of **s**, see Section 3 and Section 9.3.

Constraint: **s** $\geq$ 0.0.

6: **nx** – INTEGER

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

7: **lamda**(**nxest**) – REAL (KIND=nag_wp) array

If the warm start option is used, the values **lamda**(1), **lamda**(2), …, **lamda**(**nx**) must be left unchanged from the previous call.

8: **ny** – INTEGER

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

9: **mu**(**nyest**) – REAL (KIND=nag_wp) array

If the warm start option is used, the values **mu**(1), **mu**(2), …, **mu**(**ny**) must be left unchanged from the previous call.

10: **wrk**($lwrk$) – REAL (KIND=nag_wp) array

lwrk, the dimension of the array, must satisfy the constraint $lwrk \geq 4 \times (\mathbf{mx} + \mathbf{my}) + 11 \times (\mathbf{nxest} + \mathbf{nyest}) + \mathbf{nxest} \times \mathbf{my} + \max(\mathbf{my}, \mathbf{nxest}) + 54$.

If the warm start option is used, on entry, the values **wrk**(1), …, **wrk**(4) must be left unchanged from the previous call.

This array is used as workspace.

11: **iwrk**($liwrk$) – INTEGER array

liwrk, the dimension of the array, must satisfy the constraint $liwrk \geq 3 + \mathbf{mx} + \mathbf{my} + \mathbf{nxest} + \mathbf{nyest}$.

If the warm start option is used, on entry, the values **iwrk**(1), …, **iwrk**(3) must be left unchanged from the previous call.

This array is used as workspace.

5.2 Optional Input Parameters

1: **mx** – INTEGER

Default: the dimension of the array **x**.

m_x, the number of grid points along the x axis.

Constraint: **mx** ≥ 4.

2: **my** – INTEGER

Default: the dimension of the array **y**.

m_y, the number of grid points along the y axis.

Constraint: **my** ≥ 4.

3: **nxest** – INTEGER

4: **nyest** – INTEGER

Default: For **nxest**, the dimension of the array **lamda**. For **nyest**, the dimension of the array **mu**.

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** $= m_x/2$ and **nyest** $= m_y/2$ is sufficient. **nxest** and **nyest** never need to be larger than $m_x + 4$ and $m_y + 4$ respectively, the numbers of knots needed for interpolation (**s** $= 0.0$). See also Section 9.4.

Constraints:

nxest ≥ 8;
nyest ≥ 8.

5.3 Output Parameters

1: **nx** – INTEGER

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

2: **lamda**(**nxest**) – REAL (KIND=nag_wp) array

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

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

and

$$\textbf{lamda}(\textbf{nx} - 3) = \textbf{lamda}(\textbf{nx} - 2) = \textbf{lamda}(\textbf{nx} - 1) = \textbf{lamda}(\textbf{nx}) = \textbf{x}(\textbf{mx})$$

needed for the B-spline representation.

3: **ny** – INTEGER

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

4: **mu**(**nyest**) – REAL (KIND=nag_wp) array

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

$$\textbf{mu}(1) = \textbf{mu}(2) = \textbf{mu}(3) = \textbf{mu}(4) = \textbf{y}(1)$$

and

$$\textbf{mu}(\textbf{ny} - 3) = \textbf{mu}(\textbf{ny} - 2) = \textbf{mu}(\textbf{ny} - 1) = \textbf{mu}(\textbf{ny}) = \textbf{y}(\textbf{my})$$

needed for the B-spline representation.

5: **c**((**nxest** − **4**) × (**nyest** − **4**)) – REAL (KIND=nag_wp) array

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

6: **fp** – REAL (KIND=nag_wp)

The sum of squared residuals, θ, of the computed spline approximation. If **fp** $= 0.0$, this is an interpolating spline. **fp** should equal **s** within a relative tolerance of 0.001 unless **nx** $=$ **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: **wrk**($lwrk$) – REAL (KIND=nag_wp) array

This array is used as workspace.

8: **iwrk**($liwrk$) – INTEGER array

This array is used as workspace.

9: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

On entry, **start** $\neq$ 'C' or 'W',
or **mx** < 4,
or **my** < 4,
or **s** < 0.0,
or **s** $= 0.0$ and **nxest** $<$ **mx** $+ 4$,
or **s** $= 0.0$ and **nyest** $<$ **my** $+ 4$,
or **nxest** < 8,
or **nyest** < 8,
or $lwrk < 4 \times (\mathbf{mx} + \mathbf{my}) + 11 \times (\mathbf{nxest} + \mathbf{nyest}) + \mathbf{nxest} \times \mathbf{my} + \max(\mathbf{my}, \mathbf{nxest}) + 54$,
or $liwrk < 3 + \mathbf{mx} + \mathbf{my} + \mathbf{nxest} + \mathbf{nyest}$.

ifail $= 2$

The values of **x**(q), for $q = 1, 2, \ldots,$ **mx**, are not in strictly increasing order.

ifail $= 3$

The values of **y**(r), for $r = 1, 2, \ldots,$ **my**, are not in strictly increasing order.

ifail $= 4$

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 arguments **lamda**, **mu**, **c**, **wrk** and **iwrk**. However, if **nxest** and **nyest** are already large, say **nxest** $>$ **mx**$/2$ and **nyest** $>$ **my**$/2$, then this error exit may indicate that **s** is too small.

ifail $= 5$

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**, contact NAG.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

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

7 Accuracy

On successful exit, the approximation returned is such that its 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 nag_fit_2dspline_grid (e02dc) depends on the complexity of the shape of the data, the value of the smoothing factor S, and the number of data points. If nag_fit_2dspline_grid (e02dc) is to be called for different values of S, much time can be saved by setting **start** = 'W' after the first call.

8.2 Weighting of Data Points

nag_fit_2dspline_grid (e02dc) does not allow individual weighting of the data values. If these were determined to widely differing accuracies, it may be better to use nag_fit_2dspline_sctr (e02dd). The computation time would be very much longer, however.

8.3 Choice of s

If the standard deviation of $f_{q,r}$ is the same for all q and r (the case for which nag_fit_2dspline_grid (e02dc) is designed – see Section 9.2.) and known to be equal, at least approximately, to σ, say, then following Reinsch (1967) and choosing the argument **s** in the range $\sigma^2\left(m \pm \sqrt{2m}\right)$, where $m = m_x m_y$, is likely to give a good start in the search for a satisfactory value. If the standard deviations vary, the sum of their squares over all the data points could be used. Otherwise experimenting with different values of **s** will be required from the start, taking account of the remarks in Section 3.

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.

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 nag_fit_2dspline_grid (e02dc) 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 nag_fit_2dspline_grid (e02dc) once more with the selected value for **s** but now using **start** = 'C'. Often, nag_fit_2dspline_grid (e02dc) then returns an approximation with the same quality of fit but with fewer knots, which is therefore better if data reduction is also important.

8.4 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 nag_fit_2dspline_grid (e02dc) 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. Therefore you have 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 you want an approximation which is a simple cubic polynomial in the variable x.

8.5 Outline of Method Used

If $S = 0$, the requisite number of knots is known in advance, i.e., $n_x = m_x + 4$ and $n_y = m_y + 4$; the interior knots are located immediately as $\lambda_i = x_{i-2}$ and $\mu_j = y_{j-2}$, for $i = 5, 6, \ldots, n_x - 4$ and $j = 5, 6, \ldots, n_y - 4$. The corresponding least squares spline is then an interpolating spline and therefore a solution of the problem.

If $S > 0$, 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 $\theta > S$, new knots are added to one knot set or the other so as to reduce θ at the next stage. The new knots are located in intervals where the fit is particularly poor, their number depending on the value of S and on the progress made so far in reducing θ. Sooner or later, we find that $\theta \leq S$ and at that point the knot sets are accepted. The function then goes on to compute the (unique) spline which has these knot sets and which satisfies the full fitting criterion specified by (2) and (3). The theoretical solution has $\theta = S$. The function computes the spline by an iterative scheme which is ended when $\theta = S$ 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 nag_fit_1dspline_knots (e02ba) for least squares curve-fitting.

An exception occurs when the function finds at the start that, even with no interior knots $(n_x = n_y = 8)$, the least squares spline already has its sum of residuals $\leq S$. 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 (1982).

8.6 Evaluation of Computed Spline

The values of the computed spline at the points (x_r, y_r), for $r = 1, 2, \ldots, m$, may be obtained in the double array **ff** (see nag_fit_2dspline_evalv (e02de)), of length at least m, by the following call:

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

where $\mathtt{M} = m$ and the coordinates x_r, y_r are stored in $\mathtt{X}(k)$, $\mathtt{Y}(k)$. PX and PY have the same values as **nx** and **ny** as output from nag_fit_2dspline_grid (e02dc), and LAMDA, MU and C have the same values as **lamda**, **mu** and **c** output from nag_fit_2dspline_grid (e02dc). WRK is a double workspace array of length at least $\mathtt{PY} - 4$, and IWRK is an integer workspace array of length at least $\mathtt{PY} - 4$.

To evaluate the computed spline on a m_x by m_y rectangular grid of points in the x-y plane, which is defined by the x coordinates stored in $\mathtt{X}(q)$, for $q = 1, 2, \ldots, m_x$, and the y coordinates stored in $\mathtt{Y}(r)$, for $r = 1, 2, \ldots, m_y$, returning the results in the double array **ff** (see nag_fit_2dspline_evalm (e02df)) which is of length at least **mx** $\times$ **my**, the following call may be used:

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

where $\mathtt{MX} = m_x$, $\mathtt{MY} = m_y$. LAMDA, MU and C have the same values as **lamda**, **mu** and **c** output from nag_fit_2dspline_grid (e02dc). WRK is a double workspace array of length at least $\mathtt{LWRK} = \min(nwrk1, nwrk2)$, where $nwrk1 = \mathtt{MX} \times 4 + \mathtt{PX}$ and $nwrk2 = \mathtt{MY} \times 4 + \mathtt{PY}$. IWRK is an integer workspace array of length at least $\mathtt{LIWRK} = \mathtt{MY} + \mathtt{PY} - 4$ if $nwrk1 \geq nwrk2$, or $\mathtt{MX} + \mathtt{PX} - 4$ otherwise.

The result of the spline evaluated at grid point (q, r) is returned in element $(\mathtt{MY} \times (q - 1) + r)$ of the array FG.

9 Example

This example reads in values of **mx**, **my**, x_q, for $q = 1, 2, \ldots,$ **mx**, and y_r, for $r = 1, 2, \ldots,$ **my**, followed by values of the ordinates $f_{q,r}$ defined at the grid points $\left(x_q, y_r\right)$. It then calls nag_fit_2dspline_grid (e02dc) to compute a bicubic spline approximation for one specified value of **s**, and prints the values of the computed knots and B-spline coefficients. Finally it evaluates the spline at a small sample of points on a rectangular grid.

9.1 Program Text

```
function e02dc_example

fprintf('e02dc example results\n\n');

start = 'C';
x = [0:0.5:5];
y = [0:0.5:4];
f = [ 1         1.5      2.06     2.57     3        3.5;
      0.88758   1.3564   1.7552   2.124    2.6427   3.1715;
      0.5403    0.82045  1.0806   1.3508   1.6309   1.8611;
      0.070737  0.10611  0.15147  0.17684  0.21221  0.24458;
     -0.41515  -0.62422 -0.83229 -1.0404  -1.2484  -1.4565;
     -0.80114  -1.2317  -1.6023  -2.0029  -2.2034  -2.864;
     -0.97999  -1.485   -1.97    -2.475   -2.97    -3.265;
     -0.93446  -1.3047  -1.8729  -2.3511  -2.8094  -3.2776;
     -0.65664  -0.98547 -1.4073  -1.6741  -1.9809  -2.2878];

f(:,7:11) = [   4.04     4.5      5.04     5.505    6;
                3.5103   3.9391   4.3879   4.8367   5.2755;
                2.0612   2.4314   2.7515   2.9717   3.2418;
                0.28595  0.31632  0.35369  0.38505  0.42442;
               -1.6946  -1.8627  -2.0707  -2.2888  -2.4769;
               -3.2046  -3.6351  -4.0057  -4.4033  -4.8169;
               -3.96    -4.455   -4.97    -5.445   -5.93;
               -3.7958  -4.2141  -4.6823  -5.1405  -5.6387;
               -2.6146  -2.9314  -3.2382  -3.595   -3.9319];

% Input argument initializations
s     = 0.1;
nx    = nag_int(0); ny = nx;
mx    = size(x,2);
my    = size(y,2);
lamda = zeros(mx+4, 1);
mu    = zeros(my+4, 1);
wrk   = zeros(1000, 1);
iwrk = zeros(100, 1, nag_int_name);

% Compute bi-cubic spline for grid data
[nx, lamda, ny, mu, c, fp, wrk, iwrk, ifail] = ...
e02dc( ...
       start, x, y, f, s, nx, lamda, ny, mu, wrk, iwrk);

% Print details of spline
fprintf('\nCalling with smoothing factor S = %5.2f\n\n', s);
fprintf('Knots:   lamda      mu\n');
for j = 4:max(nx,ny)-3
  if j<=min(nx,ny)-3
    fprintf('%4d%10.4f%10.4f\n', j, lamda(j), mu(j));
  elseif j<=nx-3
    fprintf('%4d%10.4f\n', j, lamda(j));
  else
    fprintf('%4d%20.4f\n', j, mu(j));
  end
end

cp = c(1:(ny-4)*(nx-4));
cp = reshape(cp,[ny-4,nx-4]);
fprintf('\nB-spline coefficients:\n');
disp(cp);
```

```
fprintf('Weighted sum of squared residuals = %7.4f\n', fp);
if fp==0
  fprintf('(The spline is an interpolating spline)\n');
elseif nx==8 && ny==8
  fprintf('(The spline is the weighted least-squares cubic polynomial)\n');
end
fprintf('\n');

% Evaluate spline on mesh
mx = [0:0.2:5];
my = [0:0.2:4];
[ff, ifail] = e02df( ...
                    mx, my, lamda(1:nx), mu(1:ny), c);

fig1 = figure;
ff = reshape(ff,[21,26]);
meshc(mx,my,ff);
xlabel('x');
ylabel('y');
title('Least-squares bi-cubic spline fit');
```

9.2 Program Results

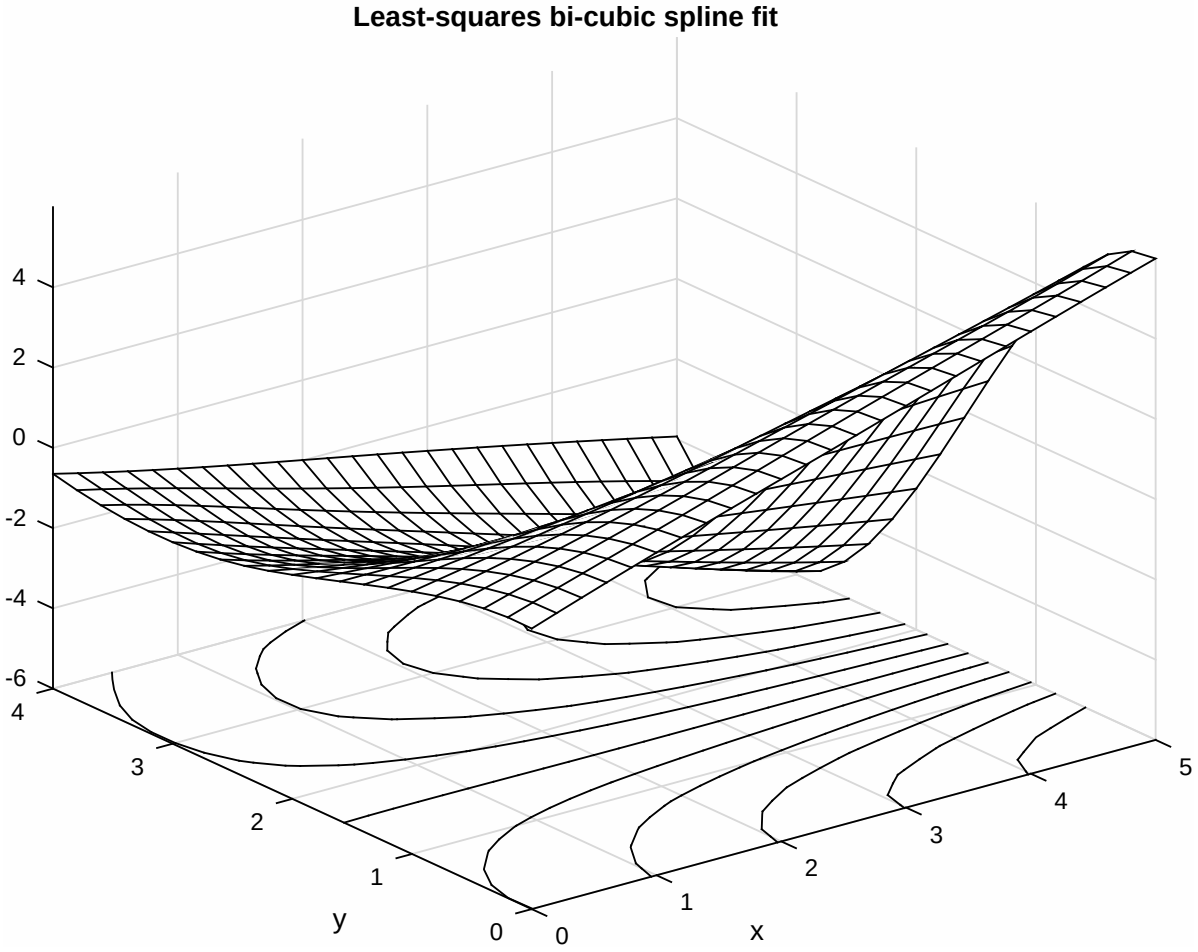
```
     e02dc example results

Calling with smoothing factor S =  0.10

Knots:   lamda      mu
   4    0.0000    0.0000
   5    1.5000    1.0000
   6    2.5000    2.0000
   7    5.0000    2.5000
   8              3.0000
   9              3.5000
  10              4.0000

B-spline coefficients:
    0.9918    1.5381    2.3913    3.9845    5.2138    5.9965
    1.0546    1.5270    2.2441    4.2217    5.0860    6.0821
    0.6098    0.9557    1.5587    2.3458    3.3860    3.7716
   -0.2915   -0.4199   -0.7399   -1.1763   -1.5527   -1.7775
   -0.8476   -1.3296   -1.8521   -3.3468   -4.3628   -5.0085
   -1.0168   -1.5952   -2.4022   -3.9390   -5.4680   -6.1656
   -0.9529   -1.3381   -2.2844   -3.9559   -5.0032   -5.8709
   -0.7711   -1.0914   -1.8488   -3.2549   -3.9444   -4.7297
   -0.6476   -1.0373   -1.5936   -2.5887   -3.3485   -3.9330

Weighted sum of squared residuals =  0.1000
```


Least-squares bi-cubic spline fit
4
2
0
-2
-4
-6
4
3
2
1
0
y
0
1
2
3
4
5
x