NAG Toolbox

nag_fit_2dspline_sctr (e02dd)

1 Purpose

nag_fit_2dspline_sctr (e02dd) computes a bicubic spline approximation to a set of scattered data. 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, rank, wrk, ifail] = nag_fit_2dspline_sctr(start, x, y, f, w, s, nx, lamda, ny, mu, wrk, 'm', m, 'nxest', nxest, 'nyest', nyest)
[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)
```

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

At Mark 22: *lwrk* was removed from the interface.

3 Description

nag_fit_2dspline_sctr (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 $[\lambda_i, \lambda_{i+1}] \times [\mu_j, \mu_{j+1}]$, 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:

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 non-negative number to be 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 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 9.2 for advice on choice of **s**.) Note however, that this function, unlike nag_fit_1dspline_auto (e02be) and nag_fit_2dspline_grid (e02dc), does not allow **s** to be set exactly to zero: to compute an interpolant to scattered data, nag_interp_2d_scat (e01sa) or nag_interp_2d_scat_shep (e01sg) should be used.

The method employed is outlined in Section 9.5 and fully described in Dierckx (1981a) and Dierckx (1981b). 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 (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** – 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** or **wrk**.

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** and **wrk** 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**(**m**) – REAL (KIND=nag_wp) array

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

4: **f**(**m**) – REAL (KIND=nag_wp) array

x(r), **y**(r), **f**(r) must be set to the coordinates 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**) – REAL (KIND=nag_wp) 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 9.4). 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** – REAL (KIND=nag_wp)

The smoothing factor, **s**.

For advice on the choice of **s**, see Section 3 and Section 9.2.

Constraint: **s** > 0.0.

7: **nx** – INTEGER

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

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

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

9: **ny** – INTEGER

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

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

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

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

lwrk, the dimension of the array, must satisfy the 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 $u =$ **nxest** $- 4$, $v =$ **nyest** $- 4$ and $w = \max(u, v)$...

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** – INTEGER

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** – INTEGER

3: **nyest** – INTEGER

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 9.3.

Constraint: **nxest** ≥ 8 and **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

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

and

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

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

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

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

and

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

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

5: **c**((**nxest** $- 4$) $\times$ (**nyest** $- 4$)) – REAL (KIND=nag_wp) array

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

6: **fp** – REAL (KIND=nag_wp)

The weighted sum of squared residuals, θ, of the computed spline approximation. **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: **rank** – INTEGER

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

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 arguments **lamda**, **mu**, **c**, **wrk** and $iwrk$. However, if **nxest** and **nyest** are already large, say $\mathbf{nxest}, \mathbf{nyest} > 4 + \sqrt{\mathbf{m}/2}$, 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**, contact 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 nag_fit_2dspline_sctr (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 $iwrk(2)$.

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** $= 3$, 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 nag_fit_2dspline_sctr (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 nag_fit_2dspline_sctr (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 argument **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 nag_fit_2dspline_sctr (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 nag_fit_2dspline_sctr (e02dd) once more with the selected value for **s** but now using **start** = 'C'. Often, nag_fit_2dspline_sctr (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 nag_fit_2dspline_sctr (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 you want 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 $[\lambda_4, \lambda_{n_x-3}] \times [\mu_4, \mu_{n_y-3}]$. 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 $\theta >$ **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 $\theta \leq$ **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 $\theta =$ **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_2dspline_panel (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 $\leq$ **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 (x_r, y_r), for $r = 1, 2, \ldots, n$, may be obtained in the double array **ff** (see nag_fit_2dspline_evalv (e02de)), of length at least n, by the following call:

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

where N $= n$ and the coordinates x_r, y_r are stored in X(k), Y(k). PX and PY have the same values as **nx** and **ny** as output from nag_fit_2dspline_sctr (e02dd), and LAMDA, MU and C have the same values as

lamda, **mu** and **c** output from nag_fit_2dspline_sctr (e02dd). WRK is a double workspace array of length at least PY − 4, and IWRK is an integer workspace array of length at least PY − 4.

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

```
[fg, ifail] = e02df(tx, ty, lamda, mu, c);
```

where KX $= k_x$, KY $= k_y$. LAMDA, MU and C have the same values as **lamda**, **mu** and **c** output from nag_fit_2dspline_sctr (e02dd). WRK is a double workspace array of length at least LWRK $= \min(nwrk1, nwrk2)$, where $nwrk1 =$ KX $\times 4 +$ PX and $nwrk2 =$ KY $\times 4 +$ PY. IWRK is an integer workspace array of length at least LIWRK $=$ KY $+$ PY $- 4$ if $nwrk1 \geq nwrk2$, or KX $+$ PX $- 4$ otherwise.

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

9 Example

This example reads in a value of **m**, followed by a set of **m** data points (x_r, y_r, f_r) and their weights w_r. It then calls nag_fit_2dspline_sctr (e02dd) 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 e02dd_example

fprintf('e02dd example results\n\n');

% Data to fit
d = [11.16   1.24  22.15;
     12.85   3.06  22.11;
     19.85  10.72   7.97;
     19.72   1.39  16.83;
     15.91   7.74  15.30;
      0.00  20.00  34.60;
     20.87  20.00   5.74;
      3.45  12.78  41.24;
     14.26  17.87  10.74;
     17.43   3.46  18.60;
     22.80  12.39   5.47;
      7.58   1.98  29.87;
     25.00  11.87   4.40;
      0.00   0.00  58.20;
      9.66  20.00   4.73;
      5.22  14.66  40.36;
     17.25  19.57   6.43;
     25.00   3.87   8.74;
     12.13  10.79  13.71;
     22.23   6.21  10.25;
     11.52   8.53  15.74;
     15.20   0.00  21.60;
      7.54  10.69  19.31;
     17.32  13.78  12.11;
      2.14  15.03  53.10;
      0.51   8.37  49.43;
     22.69  19.63   3.25;
      5.47  17.13  28.63;
     21.67  14.36   5.52;
      3.31   0.33  44.08];
x = d(:,1);
y = d(:,2);
f = d(:,3);
w = ones(size(x));
```

```
start = 'C';
s     = 10;
nx    = nag_int(0);
lamda = zeros(14,1);
ny    = nag_int(0);
mu    = zeros(14,1);
wrk   = zeros(11016, 1);
[nx, lamda, ny, mu, c, fp, rank, wrk, ifail] = ...
e02dd( ...
       start, x, y, f, w, s, nx, lamda, ny, mu, wrk);

% Print details of spline
fprintf('\nCalling with smoothing factor S = %5.2f\n', s);
fprintf('Rank deficiency = %4d\n\n',(nx-4)*(ny-4)-rank);
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 bi-cubic polynomial)\n');
end
fprintf('\n');

% Evaluate spline on mesh
mx = [3:21];
my = [2:17];
[ff, ifail] = e02df( ...
                    mx, my, lamda(1:nx), mu(1:ny), c);

fig1 = figure;
ff = reshape(ff,[16,19]);
meshc(mx,my,ff);
xlabel('x');
ylabel('y');
title('Least-squares bi-cubic spline fit of scattered data');
view(32,40);
```

9.2 Program Results

```
     e02dd example results

Calling with smoothing factor S = 10.00
Rank deficiency =    0

Knots:   lamda      mu
   4    0.0000    0.0000
   5    9.7575    9.0008
   6   18.2582   20.0000
   7   25.0000

B-spline coefficients:
   58.1559   46.3067    6.0058   31.9987    5.8554  -23.7779
   63.7813   46.7449   33.3668   18.2980   14.3600   15.9518
```

```
   40.8392  -33.7898    5.1688   13.0954   -4.1317   19.3683
   75.4362  111.9175    6.9393   17.3287    7.0928  -13.2436
   34.6068  -42.6140   25.2015   -1.9641   10.3721   -9.0871

Weighted sum of squared residuals = 10.0021
```