NAG Library Function Document

nag_2d_spline_fit_grid (e02dcc)

1 Purpose

nag_2d_spline_fit_grid (e02dcc) 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 Specification

```
#include <nag.h>
#include <nage02.h>
```

```
void nag_2d_spline_fit_grid (Nag_Start start, Integer mx, const double x[],
     Integer my, const double y[], const double f[], double s, Integer nxest,
     Integer nyest, double *fp, Nag_Comm *warmstartinf, Nag_2dSpline *spline,
     NagError *fail)
```

3 Description

nag_2d_spline_fit_grid (e02dcc) 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 $[\lambda_i, \lambda_{i+1}] \times [\mu_i, \mu_{i+1}]$, 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_1d_spline_fit (e02bec)), 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 to be specified.

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_2d_spline_eval (e02dec), nag_2d_spline_eval_rect (e02dfc) and nag_2d_spline_deriv_rect (e02dhc) 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 Arguments

1: **start** – Nag_Start *Input*

On entry: **start** must be set to **start** = Nag_Cold or Nag_Warm.

start = Nag_Cold, (cold start)
The function will build up the knot set starting with no interior knots. No values need be assigned to **spline→nx** and **spline→ny** and memory will be internally allocated to **spline→lamda**, **spline→mu**, **spline→c**, **warmstartinf→nag_w** and **warmstartinf→nag_iw**.

start = Nag_Warm (warm start)
The function will restart the knot-placing strategy using the knots found in a previous call of the function. In this case, all arguments except **s** must be unchanged from that previous call. This warm start can save much time in searching for a satisfactory value of S.

Constraint: **start** = Nag_Cold or Nag_Warm.

2: **mx** – Integer *Input*

On entry: m_x, the number of grid points along the x axis.

Constraint: **mx** ≥ 4.

3: **x**[**mx**] – const double *Input*

On entry: **x**$[q-1]$ must be set to x_q, the x coordinate of the qth grid point along the x axis, for $q = 1, 2, \dots, m_x$.

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

4: **my** – Integer *Input*

On entry: m_y, the number of grid points along the y axis.

Constraint: **my** ≥ 4.

5: **y**[**my**] – const double *Input*

On entry: **y**$[r-1]$ 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: $y_1 < y_2 < \cdots < y_{m_y}$.

6: **f**[**mx** $\times$ **my**] – const double *Input*

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

7: **s** – double *Input*

On entry: 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 9.2.

Constraint: **s** ≥ 0.0.

8: **nxest** – Integer *Input*
9: **nyest** – Integer *Input*

On entry: 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.

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

10: **fp** – double * *Output*

On exit: 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 **spline**→**nx** = **spline**→**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.

11: **warmstartinf** – Nag_Comm *

Pointer to structure of type Nag_Comm with the following members:

nag_w – double * *Input*

On entry: if the warm start option is used, the values **nag_w**[0],. . .,**nag_w**[3] must be left unchanged from the previous call.

nag_iw – Integer * *Input*

On entry: if the warm start option is used, the values **nag_iw**[0],. . .,**nag_iw**[2] must be left unchanged from the previous call.

Note that when the information contained in the pointers **nag_w** and **nag_iw** is no longer of use, or before a new call to nag_2d_spline_fit_grid (e02dcc) with the same **warmstartinf**, you should free this storage using the NAG macros `NAG_FREE`. This storage will have been allocated only if this function returns with **fail.code** = NE_NOERROR, NE_SPLINE_COEFF_CONV, or NE_-NUM_KNOTS_2D_GT_RECT.

12: **spline** – Nag_2dSpline *

Pointer to structure of type Nag_2dSpline with the following members:

nx – Integer *Input/Output*

On entry: if the warm start option is used, the value of **nx** must be left unchanged from the previous call.

On exit: the total number of knots, n_x, of the computed spline with respect to the x variable.

lamda – double * *Input/Output*

On entry: a pointer to which if **start** $=$ Nag_Cold, memory of size **nxest** is internally allocated. If the warm start option is used, the values **lamda**$[0]$, **lamda**$[1], \ldots,$ **lamda**$[$**nx**$-1]$ must be left unchanged from the previous call.

On exit: **lamda** contains the complete set of knots λ_i associated with the x variable, i.e., the interior knots **lamda**$[4]$, **lamda**$[5], \ldots,$ **lamda**$[$**nx**$-5]$ as well as the additional knots **lamda**$[0] =$ **lamda**$[1] =$ **lamda**$[2] =$ **lamda**$[3] =$ **x**$[0]$ and **lamda**$[$**nx**$-4] =$ **lamda**$[$**nx**$-3] =$ **lamda**$[$**nx**$-2] =$ **lamda**$[$**nx**$-1] =$ **x**$[$**mx**$-1]$ needed for the B-spline representation.

ny – Integer *Input/Output*

On entry: if the warm start option is used, the value of **ny** must be left unchanged from the previous call.

On exit: the total number of knots, n_y, of the computed spline with respect to the y variable.

mu – double * *Input/Output*

On entry: a pointer to which if **start** $=$ Nag_Cold, memory of size **nyest** is internally allocated. If the warm start option is used, the values **mu**$[0]$, **mu**$[1], \ldots,$ **mu**$[$**ny**$-1]$ must be left unchanged from the previous call.

On exit: **mu** contains the complete set of knots μ_i associated with the y variable, i.e., the interior knots **mu**$[4]$, **mu**$[5]$, $\ldots$, **mu**$[$**ny**$-5]$ as well as the additional knots **mu**$[0] =$ **mu**$[1] =$ **mu**$[2] =$ **mu**$[3] =$ **y**$[0]$ and **mu**$[$**ny**$-4] =$ **mu**$[$**ny**$-3] =$ **mu**$[$**ny**$-2] =$ **mu**$[$**ny**$-1] =$ **y**$[$**my**$-1]$ needed for the B-spline representation.

c – double * *Output*

On exit: a pointer to which if **start** $=$ Nag_Cold, memory of size $($**nxest**$-4) \times ($**nyest**$-4)$ is internally allocated. **c**$[(n_y - 4) \times (i-1) + j - 1]$ is the coefficient c_{ij} defined in Section 3.

Note that when the information contained in the pointers **lamda**, **mu** and **c** is no longer of use, or before a new call to nag_2d_spline_fit_grid (e02dcc) with the same **spline**, you should free this storage using the NAG macro `NAG_FREE`. This storage will have been allocated only if this function returns with **fail.code** $=$ NE_NOERROR, NE_SPLINE_COEFF_CONV, or NE_NUM_-KNOTS_2D_GT_RECT.

13: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

If the function fails with an error exit of NE_NUM_KNOTS_2D_GT_RECT or NE_SPLINE_-COEFF_CONV, then a spline approximation is returned, but it fails to satisfy the fitting criterion (see (2) and (3)) – perhaps by only a small amount, however.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument **start** had an illegal value.

NE_ENUMTYPE_WARM

start = Nag_Warm at the first call of this function. **start** must be set to**start** = Nag_Cold at the first call.

NE_INT_ARG_LT

On entry, **mx** = ⟨*value*⟩.
Constraint: **mx** ≥ 4.

On entry, **my** = ⟨*value*⟩.
Constraint: **my** ≥ 4.

On entry, **nxest** = ⟨*value*⟩.
Constraint: **nxest** ≥ 8.

On entry, **nyest** = ⟨*value*⟩.
Constraint: **nyest** ≥ 8.

NE_NOT_STRICTLY_INCREASING

The sequence **x** is not strictly increasing: **x**[⟨*value*⟩] = ⟨*value*⟩, **x**[⟨*value*⟩] = ⟨*value*⟩.
The sequence **y** is not strictly increasing: **y**[⟨*value*⟩] = ⟨*value*⟩, **y**[⟨*value*⟩] = ⟨*value*⟩.

NE_NUM_KNOTS_2D_GT_RECT

The number of knots required is greater than allowed by **nxest** or **nyest**, **nxest** = ⟨*value*⟩, **nyest** = ⟨*value*⟩. Possibly **s** is too small, especially if **nxest**, **nyest** > **mx**/2, **my**/2. **s** = ⟨*value*⟩, **mx** = ⟨*value*⟩, **my** = ⟨*value*⟩.

NE_REAL_ARG_LT

On entry, **s** must not be less than 0.0: **s** = ⟨*value*⟩.

NE_SF_D_K_CONS

On entry, **s** = ⟨*value*⟩, **nxest** = ⟨*value*⟩, **mx** = ⟨*value*⟩.
Constraint: **nxest** $\geq$ **mx** $+ 4$ when **s** $= 0.0$.

On entry, **s** = ⟨*value*⟩, **nyest** = ⟨*value*⟩, **my** = ⟨*value*⟩.
Constraint: **nyest** $\geq$ **mx** $+ 4$ when **s** $= 0.0$.

NE_SPLINE_COEFF_CONV

The iterative process has failed to converge. Possibly **s** is too small: **s** = ⟨*value*⟩.

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 Parallelism and Performance

nag_2d_spline_fit_grid (e02dcc) is not threaded in any implementation.

9 Further Comments

9.1 Timing

The time taken for a call of nag_2d_spline_fit_grid (e02dcc) 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_2d_spline_fit_grid (e02dcc) is to be called for different values of S, much time can be saved by setting **start** $=$ Nag_Warm after the first call.

9.2 Weighting of Data Points

nag_2d_spline_fit_grid (e02dcc) does not allow individual weighting of the data values. If these were determined to widely differing accuracies, it may be better to use nag_2d_spline_fit_scat (e02ddc). The computation time would be very much longer, however.

9.3 Choice of s

If the standard deviation of $f_{q,r}$ is the same for all q and r (the case for which this function is designed – see Section 9.2) and known to be equal, at least approximately, to σ, say, then following Reinsch (1967) and choosing the smoothing factor 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., $S = \mathbf{fp}_0/10$, $S = \mathbf{fp}_0/100$, 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_2d_spline_fit_grid (e02dcc) is called with **start** $=$ Nag_Warm, 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** $=$ Nag_Warm, a fit can finally be accepted as satisfactory, it may be worthwhile to call nag_2d_spline_fit_grid (e02dcc) once more with the selected value for S but now using **start** $=$ Nag_Cold. Often, nag_2d_spline_fit_grid (e02dcc) then returns an approximation with the same quality of fit but with fewer knots, which is therefore better if data reduction is also important.

9.4 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_2d_spline_fit_grid (e02dcc) 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.

9.5 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_1d_spline_ fit_knots (e02bac) for least squares curve fitting.

An exception occurs when the function finds at the start that, even with no interior knots $\left(n_x = n_y = 8\right)$, 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).

9.6 Evaluation of Computed Spline

The values of the computed spline at the points $(\mathbf{tx}(r-1), \mathbf{ty}(r-1))$, for $r = 1, 2, \ldots, \mathbf{n}$, may be obtained in the array **ff**, of length at least **n**, by the following code:

```
e02dec(n, tx, ty, ff, &spline, &fail)
```

where **spline** is a structure of type Nag_2dSpline which is an output argument of nag_2d_spline_fit_ grid (e02dcc).

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 coordinates stored in $\mathbf{tx}(q-1)$, for $q = 1, 2, \ldots, \mathbf{kx}$, and the y coordinates stored in $\mathbf{ty}(r-1)$, for $r = 1, 2, \ldots, \mathbf{ky}$, returning the results in the array **fg** which is of length at least $\mathbf{kx} \times \mathbf{ky}$, the following call may be used:

```
e02dfc(kx, ky, tx, ty, fg, &spline, &fail)
```

where **spline** is a structure of type Nag_2dSpline which is an output argument of nag_2d_spline_fit_ grid (e02dcc). The result of the spline evaluated at grid point (q, r) is returned in element $[\mathbf{ky} \times (q-1) + r - 1]$ of the array **fg**.

10 Example

This example program reads in values of **mx**, **my**, x_q, for $q = 1, 2, \ldots, \mathbf{mx}$, and y_r, for $r = 1, 2, \ldots, \mathbf{my}$, followed by values of the ordinates $f_{q,r}$ defined at the grid points $\left(x_q, y_r\right)$. It then calls nag_2d_spline_fit_grid (e02dcc) 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.

10.1 Program Text

```
/* nag_2d_spline_fit_grid (e02dcc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 *
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nage02.h>

int main(void)
{
  Integer exit_status = 0, i, j, mx, my, npx, npy, nx, ny;
  Nag_2dSpline spline;
  Nag_Comm warmstartinf;
  Nag_Start start;
  double delta, *f = 0, *fg = 0, fp, *px = 0, *py = 0, s, *x = 0, xhi,
         xlo, *y = 0, yhi;
  double ylo;
  NagError fail;

  INIT_FAIL(fail);

  /* Initialize spline */
  spline.lamda = 0;
  spline.mu = 0;
  spline.c = 0;

  warmstartinf.nag_w = 0;
  warmstartinf.nag_iw = 0;

  printf("nag_2d_spline_fit_grid (e02dcc) Example Program Results\n");
#ifdef _WIN32
  scanf_s("%*[^\n]"); /* Skip heading in data file */
#else
  scanf("%*[^\n]"); /* Skip heading in data file */
#endif
  /* Input the number of x, y co-ordinates mx, my. */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "", &mx, &my);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "", &mx, &my);
#endif

  if (mx >= 4 && my >= 4) {
    if (!(f = NAG_ALLOC(mx * my, double)) ||
        !(x = NAG_ALLOC(mx, double)) || !(y = NAG_ALLOC(my, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  }
  else {
    printf("Invalid mx or my.\n");
    exit_status = 1;
    return exit_status;
  }
  /* Input the x co-ordinates followed by the y co-ordinates. */
  for (i = 0; i < mx; i++)
#ifdef _WIN32
    scanf_s("%lf", &x[i]);
#else
    scanf("%lf", &x[i]);
#endif
  for (i = 0; i < my; i++)
#ifdef _WIN32
    scanf_s("%lf", &y[i]);
#else
    scanf("%lf", &y[i]);
#endif
  /* Input the mx*my function values f at the grid points. */
  for (i = 0; i < mx * my; i++)
#ifdef _WIN32
    scanf_s("%lf", &f[i]);
#else
    scanf("%lf", &f[i]);
#endif
  start = Nag_Cold;
#ifdef _WIN32
  scanf_s("%lf", &s);
#else
  scanf("%lf", &s);
#endif
  /* Determine the spline approximation. */
  /* nag_2d_spline_fit_grid (e02dcc).
   * Least squares bicubic spline fit with automatic knot
   * placement, two variables (rectangular grid)
   */
  nag_2d_spline_fit_grid(start, mx, x, my, y, f, s, mx + 4, my + 4,
                         &fp, &warmstartinf, &spline, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_2d_spline_fit_grid (e02dcc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  nx = spline.nx;
  ny = spline.ny;

  printf("\nCalling with smoothing factor s = %13.4e:"
         " spline.nx = %2" NAG_IFMT ", spline.ny = %2" NAG_IFMT ".\n\n",
         s, nx, ny);
  /* Print the knot sets, lamda and mu. */
  printf("Distinct knots in x direction located  at\n");
  for (j = 3; j < spline.nx - 3; j++)
    printf("%12.4f%s", spline.lamda[j],
           ((j - 3) % 5 == 4 || j == spline.nx - 4) ? "\n" : " ");
  printf("\nDistinct knots in y direction located  at\n");
  for (j = 3; j < spline.ny - 3; j++)
    printf("%12.4f%s", spline.mu[j], ((j - 3) % 5 == 4 || j == spline.ny - 4)
           ? "\n" : " ");
  printf("\nThe B-spline coefficients:\n\n");
  for (i = 0; i < ny - 4; i++) {
    for (j = 0; j < nx - 4; j++)
      printf("%9.4f", spline.c[i + j * (ny - 4)]);
    printf("\n");
  }
  printf("\nSum of squared residuals fp = %13.4e\n", fp);
  if (fp == 0.0)
    printf("\nThe spline is an interpolating spline\n");
  else if (nx == 8 && ny == 8)
    printf("\nThe spline is the least squares bi-cubic polynomial\n");

  /* Evaluate the spline on a rectangular grid at npx*npy points
   * over the domain (xlo to xhi) x (ylo to yhi).
   */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%lf%lf", &npx, &xlo, &xhi);
#else
  scanf("%" NAG_IFMT "%lf%lf", &npx, &xlo, &xhi);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%lf%lf", &npy, &ylo, &yhi);
#else
  scanf("%" NAG_IFMT "%lf%lf", &npy, &ylo, &yhi);
#endif
  if (npx >= 1 && npy >= 1) {
    if (!(fg = NAG_ALLOC(npx * npy, double)) ||
        !(px = NAG_ALLOC(npx, double)) || !(py = NAG_ALLOC(npy, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  }
  else {
    printf("Invalid npx or npy.\n");
    exit_status = 1;
    return exit_status;
  }
  delta = (xhi - xlo) / (npx - 1);
  for (i = 0; i < npx; i++)
    px[i] = MIN(xlo + i * delta, xhi);
  for (i = 0; i < npy; i++)
    py[i] = MIN(ylo + i * delta, yhi);
  /* nag_2d_spline_eval_rect (e02dfc).
   * Evaluation of bicubic spline, at a mesh of points
   */
  nag_2d_spline_eval_rect(npx, npy, px, py, fg, &spline, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_2d_spline_eval_rect (e02dfc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  printf("\nValues of computed spline:\n");
  printf("\n          x");
  for (i = 0; i < npx; i++)
    printf("%7.2f  ", px[i]);
  printf("\n     y\n");
  for (i = npy - 1; i >= 0; i--) {
    printf("%7.2f   ", py[i]);
    for (j = 0; j < npx; ++j)
      printf("%8.2f ", fg[npy * j + i]);
    printf("\n");
  }
END:
  NAG_FREE(spline.lamda);
  NAG_FREE(spline.mu);
  NAG_FREE(spline.c);
  NAG_FREE(warmstartinf.nag_w);
  NAG_FREE(warmstartinf.nag_iw);
  NAG_FREE(f);
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(fg);
  NAG_FREE(px);
  NAG_FREE(py);
  return exit_status;
}
```

10.2 Program Data

```
nag_2d_spline_fit_grid (e02dcc) Example Program Data
 11    9
 0.0000E+00  5.0000E-01  1.0000E+00  1.5000E+00  2.0000E+00
 2.5000E+00  3.0000E+00  3.5000E+00  4.0000E+00  4.5000E+00
 5.0000E+00
 0.0000E+00  5.0000E-01  1.0000E+00  1.5000E+00  2.0000E+00
 2.5000E+00  3.0000E+00  3.5000E+00  4.0000E+00
 1.0000E+00  8.8758E-01  5.4030E-01  7.0737E-02 -4.1515E-01
-8.0114E-01 -9.7999E-01 -9.3446E-01 -6.5664E-01  1.5000E+00
 1.3564E+00  8.2045E-01  1.0611E-01 -6.2422E-01 -1.2317E+00
-1.4850E+00 -1.3047E+00 -9.8547E-01  2.0600E+00  1.7552E+00
 1.0806E+00  1.5147E-01 -8.3229E-01 -1.6023E+00 -1.9700E+00
-1.8729E+00 -1.4073E+00  2.5700E+00  2.1240E+00  1.3508E+00
 1.7684E-01 -1.0404E+00 -2.0029E+00 -2.4750E+00 -2.3511E+00
-1.6741E+00  3.0000E+00  2.6427E+00  1.6309E+00  2.1221E-01
-1.2484E+00 -2.2034E+00 -2.9700E+00 -2.8094E+00 -1.9809E+00
 3.5000E+00  3.1715E+00  1.8611E+00  2.4458E-01 -1.4565E+00
-2.8640E+00 -3.2650E+00 -3.2776E+00 -2.2878E+00  4.0400E+00
 3.5103E+00  2.0612E+00  2.8595E-01 -1.6946E+00 -3.2046E+00
-3.9600E+00 -3.7958E+00 -2.6146E+00  4.5000E+00  3.9391E+00
 2.4314E+00  3.1632E-01 -1.8627E+00 -3.6351E+00 -4.4550E+00
-4.2141E+00 -2.9314E+00  5.0400E+00  4.3879E+00  2.7515E+00
 3.5369E-01 -2.0707E+00 -4.0057E+00 -4.9700E+00 -4.6823E+00
-3.2382E+00  5.5050E+00  4.8367E+00  2.9717E+00  3.8505E-01
-2.2888E+00 -4.4033E+00 -5.4450E+00 -5.1405E+00 -3.5950E+00
 6.0000E+00  5.2755E+00  3.2418E+00  4.2442E-01 -2.4769E+00
-4.8169E+00 -5.9300E+00 -5.6387E+00 -3.9319E+00
 0.1
 6    0.0   5.0
 5    0.0   4.0
```

10.3 Program Results

```
nag_2d_spline_fit_grid (e02dcc) Example Program Results

Calling with smoothing factor s =    1.0000e-01: spline.nx = 10, spline.ny = 13.

Distinct knots in x direction located  at
      0.0000       1.5000       2.5000       5.0000

Distinct knots in y direction located  at
      0.0000       1.0000       2.0000       2.5000       3.0000
      3.5000       4.0000

The 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

Sum of squared residuals fp =    1.0004e-01

Values of computed spline:

          x   0.00     1.00     2.00     3.00     4.00     5.00
    y
   4.00      -0.65    -1.36    -1.99    -2.61    -3.25    -3.93
   3.00      -0.98    -1.97    -2.91    -3.91    -4.97    -5.92
   2.00      -0.42    -0.83    -1.24    -1.66    -2.08    -2.48
   1.00       0.54     1.09     1.61     2.14     2.71     3.24
   0.00       0.99     2.04     3.03     4.01     5.02     6.00
```