NAG Library Function Document

nag_2d_spline_fit_scat (e02ddc)

1 Purpose

nag_2d_spline_fit_scat (e02ddc) 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 Specification

```
#include <nag.h>
#include <nage02.h>
void nag_2d_spline_fit_scat (Nag_Start start, Integer m, const double x[],
     const double y[], const double f[], const double weights[], double s,
     Integer nxest, Integer nyest, double *fp, Integer *rank,
     double *warmstartinf, Nag_2dSpline *spline, NagError *fail)
```

3 Description

nag_2d_spline_fit_scat (e02ddc) 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 $[\lambda_i, \lambda_{i+1}] \times [\mu_i, \mu_{i+1}]$, 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:

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 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 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.) Note however, that this function, unlike nag_1d_spline_fit (e02bec) and nag_2d_spline_fit_grid (e02dcc), does not allow S to be set exactly to zero.

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_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 (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 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** and **spline→c**.

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: **m** – Integer *Input*

On entry: m, the number of data points.

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

3: **x**[**m**] – const double *Input*
4: **y**[**m**] – const double *Input*
5: **f**[**m**] – const double *Input*

On entry: **x**$[r-1]$, **y**$[r-1]$, **f**$[r-1]$ 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.

6: **weights**[**m**] – const double *Input*

On entry: **weights**$[r-1]$ 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 the e02 Chapter Introduction.

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

7: **s** – double *Input*

On entry: the smoothing factor, S. 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** = **nyest** $= 5 + \sqrt{m}$ is sufficient. See also Section 9.3.

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

10: **fp** – double * *Output*

On exit: the weighted sum of squared residuals, θ, of the computed spline approximation. **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: **rank** – Integer * *Output*

On exit: **rank** gives the rank of the system of equations used to compute the final spline (as determined by a suitable machine-dependent threshold). When **rank** = (**spline**→**nx** $-$ 4) $\times$ (**spline**→**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.

12: **warmstartinf** – double * *Output*

On exit: if the warm start option is used, its value must be left unchanged from the previous call.

13: **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**$[3] = x_{\min}$ and

lamda[**nx** − 4] = **lamda**[**nx** − 3] = **lamda**[**nx** − 2] = **lamda**[**nx** − 1] = $x_{\max}$ needed for the B-spline representation (where $x_{\min}$ and $x_{\max}$ are as described in Section 3).

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], ..., **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], ..., **mu**[**ny** − 5] as well as the additional knots **mu**[0] = **mu**[1] = **mu**[2] = **mu**[3] = $y_{\min}$ and **mu**[**ny** − 4] = **mu**[**ny** − 3] = **mu**[**ny** − 2] = **mu**[**ny** − 1] = $y_{\max}$ needed for the B-spline representation (where $y_{\min}$ and $y_{\max}$ are as described in Section 3).

c – double * *Output*

On exit: a pointer to which, if **start** = Nag_Cold, memory of size (**nxest** − 4) × (**nyest** − 4) is internally allocated. **c**[$\big(n_y - 4\big) \times (i-1) + j - 1$] 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_scat (e02ddc) 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_NUM_KNOTS_2D_GT_SCAT, NE_NUM_COEFF_GT, NE_NO_ADDITIONAL_KNOTS, NE_SPLINE_COEFF_CONV or, possibly, NE_ALLOC_FAIL.

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

If the function fails with an error exit of NE_NUM_KNOTS_2D_GT_SCAT, NE_NUM_COEFF_GT, NE_NO_ADDITIONAL_KNOTS 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_ALL_ELEMENTS_EQUAL

On entry, all the values in the array **x** must not be equal.

On entry, all the values in the array **y** must not be equal.

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, **nxest** $= \langle value \rangle$.
Constraint: **nxest** ≥ 8.

On entry, **nyest** $= \langle value \rangle$.
Constraint: **nyest** ≥ 8.

NE_NO_ADDITIONAL_KNOTS

No more knots added; the additional knot would coincide with an old one. Possibly an inaccurate data point has too large a weight, or **s** is too small. **s** $= \langle value \rangle$.

NE_NON_ZERO_WEIGHTS

On entry, the number of data points with nonzero weights $= \langle value \rangle$.
Constraint: the number of nonzero weights ≥ 16.

NE_NUM_COEFF_GT

No more knots can be added because the number of B-spline coefficients already exceeds **m**. Either **m** or **s** is probably too small: **m** $= \langle value \rangle$, **s** $= \langle value \rangle$.

NE_NUM_KNOTS_2D_GT_SCAT

The number of knots required is greater than allowed by **nxest** or **nyest**, **nxest** $= \langle value \rangle$, **nyest** $= \langle value \rangle$. Possibly **s** is too small, especially if **nxest**, **nyest** $> 5 + \sqrt{\mathbf{m}}$. **s** $= \langle value \rangle$, **m** $= \langle value \rangle$.

NE_REAL_ARG_LE

On entry, **s** must not be less than or equal to 0.0: **s** $= \langle value \rangle$.

NE_SPLINE_COEFF_CONV

The iterative process has failed to converge. Possibly **s** is too small: **s** $= \langle value \rangle$.

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 Parallelism and Performance

Not applicable.

9 Further Comments

9.1 Timing

The time taken for a call of nag_2d_spline_fit_scat (e02ddc) 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_scat (e02ddc) is to be called for different values of S, much time can be saved by setting **start** = Nag_Warm after the first call.

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

9.2 Choice of *S*

If the weights have been correctly chosen (see 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., $S = \mathbf{fp}_0/10$, $S = \mathbf{fp}_0/100$, 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_2d_spline_fit_scat (e02ddc) 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_scat (e02ddc) once more with the selected value for S but now using **start** = Nag_Cold. Often, nag_2d_spline_fit_scat (e02ddc) then returns an approximation with the same quality of fit but with fewer knots, which is therefore better if data reduction is also important.

9.3 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_2d_spline_fit_scat (e02ddc) 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.

9.4 Restriction of the Approximation Domain

The fit obtained is not defined outside the rectangle $[\lambda_4, \lambda_{n_x-3}] \times \left[\mu_4, \mu_{n_y-3}\right]$. 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.

9.5 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. 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 $\left(n_x = n_y = 8\right)$, 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).

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_scat (e02ddc).

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_scat (e02ddc). 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 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_2d_spline_fit_scat (e02ddc) 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_scat (e02ddc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 *
 * Mark 6 revised, 2000.
 * Mark 8 revised, 2004.
 */

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

int main(void)
{

  Integer      exit_status = 0, i, j, m, npx, npy, nx, nxest, ny, nyest, rank;
  NagError     fail;
  Nag_2dSpline spline;
  Nag_Start    start;
  double       delta, *f = 0, *fg = 0, fp, *px = 0, *py = 0, s, warmstartinf;
  double       *weights = 0, *x = 0, xhi, xlo, *y = 0, yhi, ylo;

  INIT_FAIL(fail);

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

  nxest = 14;
  nyest = 14;
  printf("nag_2d_spline_fit_scat (e02ddc) 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 data-points m. */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"", &m);
#else
  scanf("%"NAG_IFMT"", &m);
#endif
  if (m >= 16)
    {
      if (!(f = NAG_ALLOC(m, double)) ||
          !(weights = NAG_ALLOC(m, double)) ||
          !(x = NAG_ALLOC(m, double)) ||
          !(y = NAG_ALLOC(m, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  else
    {
      printf("Invalid m.\n");
      exit_status = 1;
      return exit_status;
    }
  /* Input the data-points and the weights. */
  for (i = 0; i < m;  i++)
#ifdef _WIN32
    scanf_s("%lf%lf%lf%lf", &x[i], &y[i], &f[i], &weights[i]);
#else
    scanf("%lf%lf%lf%lf", &x[i], &y[i], &f[i], &weights[i]);
#endif
  start = Nag_Cold;
#ifdef _WIN32
  if (scanf_s("%lf", &s) != EOF)
    {
      /* Determine the spline approximation. */

      /* nag_2d_spline_fit_scat (e02ddc).
       * Least-squares bicubic spline fit with automatic knot
       * placement, two variables (scattered data)
       */
      nag_2d_spline_fit_scat(start, m, x, y, f, weights, s, nxest, nyest, &fp,
                             &rank, &warmstartinf, &spline, &fail);
#else
  if (scanf("%lf", &s) != EOF)
    {
      /* Determine the spline approximation. */

      /* nag_2d_spline_fit_scat (e02ddc).
       * Least-squares bicubic spline fit with automatic knot
       * placement, two variables (scattered data)
       */
      nag_2d_spline_fit_scat(start, m, x, y, f, weights, s, nxest, nyest, &fp,
                             &rank, &warmstartinf, &spline, &fail);
#endif
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_2d_spline_fit_scat (e02ddc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }
      nx = spline.nx;
      ny = spline.ny;
      printf("\nCalling with smoothing factor s = %13.4e, nx = %1"NAG_IFMT","
              " ny = %1"NAG_IFMT"\n", s, nx, ny);
```

```
      printf("rank deficiency = %1"NAG_IFMT"\n\n", (nx-4)*(ny-4)-rank);

      /* 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("%9.2f", spline.c[i+j*(ny-4)]);
          printf("\n");
        }

      printf("\n Sum of squared residuals fp = %13.4e\n", fp);
      if (nx == 8 && ny == 8)
        printf("The 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(xhi, xlo+i*delta);
      for (i = 0; i < npy;  i++)
        py[i] = MIN(yhi, ylo+i*delta);

      /* 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\n");
      printf("          x");
      for (i = 0; i < npx; i++)
        printf("%8.2f ", px[i]);
      printf("\n     y\n");
      for (i = npy-1; i >= 0; i--)
        {
          printf("%8.2f   ", py[i]);
          for (j = 0; j < npx; j++)
            printf("%8.2f ", fg[npy*j+i]);
          printf("\n");
        }
      /* Free memory used by spline */
      NAG_FREE(spline.lamda);
      NAG_FREE(spline.mu);
      NAG_FREE(spline.c);
      NAG_FREE(fg);
      NAG_FREE(px);
      NAG_FREE(py);
    }
 END:
  NAG_FREE(f);
  NAG_FREE(weights);
  NAG_FREE(x);
  NAG_FREE(y);
  return exit_status;
}
```

10.2 Program Data

```
nag_2d_spline_fit_scat (e02ddc) Example Program Data
 30
 11.16   1.24  22.15  1.00
 12.85   3.06  22.11  1.00
 19.85  10.72   7.97  1.00
 19.72   1.39  16.83  1.00
 15.91   7.74  15.30  1.00
  0.00  20.00  34.60  1.00
 20.87  20.00   5.74  1.00
  3.45  12.78  41.24  1.00
 14.26  17.87  10.74  1.00
 17.43   3.46  18.60  1.00
 22.80  12.39   5.47  1.00
  7.58   1.98  29.87  1.00
 25.00  11.87   4.40  1.00
  0.00   0.00  58.20  1.00
  9.66  20.00   4.73  1.00
  5.22  14.66  40.36  1.00
 17.25  19.57   6.43  1.00
 25.00   3.87   8.74  1.00
 12.13  10.79  13.71  1.00
 22.23   6.21  10.25  1.00
 11.52   8.53  15.74  1.00
 15.20   0.00  21.60  1.00
  7.54  10.69  19.31  1.00
 17.32  13.78  12.11  1.00
  2.14  15.03  53.10  1.00
  0.51   8.37  49.43  1.00
 22.69  19.63   3.25  1.00
  5.47  17.13  28.63  1.00
 21.67  14.36   5.52  1.00
  3.31   0.33  44.08  1.00
 10.0
   7    3.0   21.0
   6    2.0   17.0
```

10.3 Program Results

```
nag_2d_spline_fit_scat (e02ddc) Example Program Results

Calling with smoothing factor s =  1.0000e+01, nx = 10, ny = 9
rank deficiency = 0

Distinct knots in x direction located  at
      0.0000       9.7575      18.2582      25.0000

Distinct knots in y direction located  at
      0.0000       9.0008      20.0000

The B-spline coefficients:

    58.16    46.31     6.01    32.00     5.86   -23.78
    63.78    46.74    33.37    18.30    14.36    15.95
    40.84   -33.79     5.17    13.10    -4.13    19.37
    75.44   111.92     6.94    17.33     7.09   -13.24
    34.61   -42.61    25.20    -1.96    10.37    -9.09

 Sum of squared residuals fp =  1.0002e+01

Values of computed spline:

          x    3.00     6.00     9.00    12.00    15.00    18.00    21.00
     y
 17.00       40.74    28.62    19.84    14.29    11.21     9.46     7.09
 14.00       48.34    33.97    21.56    14.71    12.32    10.82     7.15
 11.00       37.26    24.46    17.21    14.14    13.02    11.23     7.29
  8.00       30.25    19.66    16.90    16.28    15.21    12.71     8.99
  5.00       36.64    26.75    23.07    21.13    18.97    15.90    11.98
  2.00       45.04    33.70    26.25    22.88    21.62    19.39    13.40
```