NAG Library Function Document

nag_2d_spline_fit_panel (e02dac)

1 Purpose

nag_2d_spline_fit_panel (e02dac) forms a minimal, weighted least squares bicubic spline surface fit with prescribed knots to a given set of data points.

2 Specification

```
#include <nag.h>
#include <nage02.h>
void nag_2d_spline_fit_panel (Integer m, const double x[], const double y[],
     const double f[], const double w[], const Integer point[], double dl[],
     double eps, double *sigma, Integer *rank, Nag_2dSpline *spline,
     NagError *fail)
```

3 Description

nag_2d_spline_fit_panel (e02dac) determines a bicubic spline fit $s(x,y)$ to the set of data points (x_r, y_r, f_r) with weights w_r, for $r = 1, 2, \ldots, m$. The two sets of internal knots of the spline, $\{\lambda\}$ and $\{\mu\}$, associated with the variables x and y respectively, are prescribed by you. These knots can be thought of as dividing the data region of the (x,y) plane into panels (see Figure 1 in Section 5). A bicubic spline consists of a separate bicubic polynomial in each panel, the polynomials joining together with continuity up to the second derivative across the panel boundaries.

$s(x,y)$ has the property that Σ, the sum of squares of its weighted residuals ρ_r, for $r = 1, 2, \ldots, m$, where

$$\rho_r = w_r(s(x_r, y_r) - f_r) \tag{1}$$

is as small as possible for a bicubic spline with the given knot sets. The function produces this minimized value of Σ and the coefficients c_{ij} in the B-spline representation of $s(x,y)$ – see Section 9. nag_2d_spline_eval (e02dec), nag_2d_spline_eval_rect (e02dfc) and nag_2d_spline_deriv_rect (e02dhc) are available to compute values and derivatives of the fitted spline from the coefficients c_{ij}.

The least squares criterion is not always sufficient to determine the bicubic spline uniquely: there may be a whole family of splines which have the same minimum sum of squares. In these cases, the function selects from this family the spline for which the sum of squares of the coefficients c_{ij} is smallest: in other words, the minimal least squares solution. This choice, although arbitrary, reduces the risk of unwanted fluctuations in the spline fit. The method employed involves forming a system of m linear equations in the coefficients c_{ij} and then computing its least squares solution, which will be the minimal least squares solution when appropriate. The basis of the method is described in Hayes and Halliday (1974). The matrix of the equation is formed using a recurrence relation for B-splines which is numerically stable (see Cox (1972) and de Boor (1972) – the former contains the more elementary derivation but, unlike de Boor (1972), does not cover the case of coincident knots). The least squares solution is also obtained in a stable manner by using orthogonal transformations, viz. a variant of Givens rotation (see Gentleman (1973)). This requires only one row of the matrix to be stored at a time. Advantage is taken of the stepped-band structure which the matrix possesses when the data points are suitably ordered, there being at most sixteen nonzero elements in any row because of the definition of B-splines. First the matrix is reduced to upper triangular form and then the diagonal elements of this triangle are examined in turn. When an element is encountered whose square, divided by the mean squared weight, is less than a threshold ϵ, it is replaced by zero and the rest of the elements in its row are reduced to zero by rotations with the remaining rows. The rank of the system is taken to be the number of nonzero diagonal elements in the final triangle, and the nonzero rows of this triangle are used to compute the minimal least squares solution. If all the diagonal elements are nonzero, the rank is

equal to the number of coefficients c_{ij} and the solution obtained is the ordinary least squares solution, which is unique in this case.

4 References

Cox M G (1972) The numerical evaluation of B-splines *J. Inst. Math. Appl.* **10** 134–149

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

Gentleman W M (1973) Least squares computations by Givens transformations without square roots *J. Inst. Math. Applic.* **12** 329–336

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

5 Arguments

1: **m** – Integer *Input*

On entry: m, the number of data points.

Constraint: **m** > 1.

2: **x**[**m**] – const double *Input*

3: **y**[**m**] – const double *Input*

4: **f**[**m**] – const double *Input*

On entry: the coordinates of the data point (x_r, y_r, f_r), for $r = 1, 2, \ldots, m$. The order of the data points is immaterial, but see the array **point**.

5: **w**[**m**] – const double *Input*

On entry: the weight w_r of the rth data point. It is important to note the definition of weight implied by the equation (1) in Section 3, since it is also common usage to define weight as the square of this weight. In this function, each w_r should be chosen inversely proportional to the (absolute) accuracy of the corresponding f_r, as expressed, for example, by the standard deviation or probable error of the f_r. When the f_r are all of the same accuracy, all the w_r may be set equal to 1.0.

6: **point**[dim] – const Integer *Input*

Note: the dimension, dim, of the array **point** must be at least (**m** + (**spline**.**nx** − 7) × (**spline**.**ny** − 7)).

On entry: indexing information usually provided by nag_2d_panel_sort (e02zac) which enables the data points to be accessed in the order which produces the advantageous matrix structure mentioned in Section 3. This order is such that, if the (x, y) plane is thought of as being divided into rectangular panels by the two sets of knots, all data in a panel occur before data in succeeding panels, where the panels are numbered from bottom to top and then left to right with the usual arrangement of axes, as indicated in Figure 1.

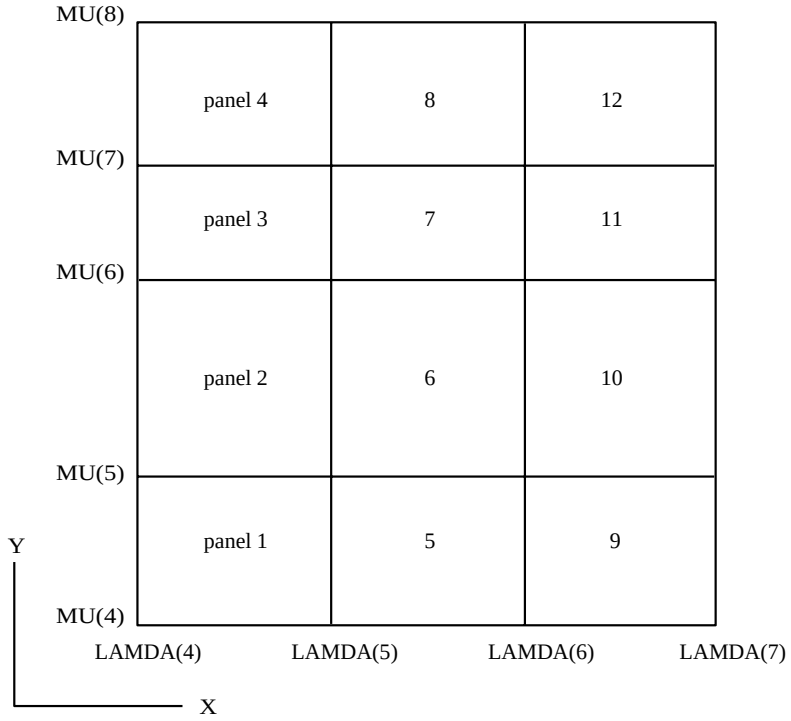

Figure 1

A data point lying exactly on one or more panel sides is considered to be in the highest numbered panel adjacent to the point. nag_2d_panel_sort (e02zac) should be called to obtain the array **point**, unless it is provided by other means.

7: **dl**$[dim]$ – double *Output*

Note: the dimension, *dim*, of the array **dl** must be at least $(\textbf{spline.nx} - 4) \times (\textbf{spline.ny} - 4)$.

On exit: gives the squares of the diagonal elements of the reduced triangular matrix, divided by the mean squared weight. It includes those elements, less than ϵ, which are treated as zero (see Section 3).

8: **eps** – double *Input*

On entry: a threshold ϵ for determining the effective rank of the system of linear equations. The rank is determined as the number of elements of the array **dl** which are nonzero. An element of **dl** is regarded as zero if it is less than ϵ. ***Machine precision*** is a suitable value for ϵ in most practical applications which have only 2 or 3 decimals accurate in data. If some coefficients of the fit prove to be very large compared with the data ordinates, this suggests that ϵ should be increased so as to decrease the rank. The array **dl** will give a guide to appropriate values of ϵ to achieve this, as well as to the choice of ϵ in other cases where some experimentation may be needed to determine a value which leads to a satisfactory fit.

9: **sigma** – double * *Output*

On exit: Σ, the weighted sum of squares of residuals. This is not computed from the individual residuals but from the right-hand sides of the orthogonally-transformed linear equations. For further details see page 97 of Hayes and Halliday (1974). The two methods of computation are theoretically equivalent, but the results may differ because of rounding error.

10: **rank** – Integer * *Output*

On exit: the rank of the system as determined by the value of the threshold ϵ.

rank $= (\mathbf{spline.nx} - 4) \times (\mathbf{spline.ny} - 4)$
The least squares solution is unique.

rank $\neq (\mathbf{spline.nx} - 4) \times (\mathbf{spline.ny} - 4)$
The minimal least squares solution is computed.

11: **spline** – Nag_2dSpline *

Pointer to structure of type Nag_2dSpline with the following members:

nx – Integer *Input*

On entry: **nx** must specify the total number of knots associated with the variables x. It is such that $\mathbf{nx} - 8$ is the number of interior knots.

Constraint: $\mathbf{nx} \geq 8$.

lamda – double *Input/Output*

On entry: $\mathbf{lamda}[i + 4]$ must contain the ith interior knot λ_{i+4} associated with the variable x, for $i = 1, 2, \ldots \mathbf{nx} - 8$. The knots must be in nondecreasing order and lie strictly within the range covered by the data values of x. A knot is a value of x at which the spline is allowed to be discontinuous in the third derivative with respect to x, though continuous up to the second derivative. This degree of continuity can be reduced, if you require, by the use of coincident knots, provided that no more than four knots are chosen to coincide at any point. Two, or three, coincident knots allow loss of continuity in, respectively, the second and first derivative with respect to x at the value of x at which they coincide. Four coincident knots split the spline surface into two independent parts. For choice of knots see Section 9.

On exit: the interior knots $\mathbf{lamda}[4]$ to $\mathbf{lamda}[\mathbf{nx} - 5]$ are unchanged, and the segments $\mathbf{LAMDA}(1 : 4)$ and $\mathbf{LAMDA}(\mathbf{nx} - 3 : \mathbf{nx})$ contain additional (exterior) knots introduced by the function in order to define the full set of B-splines required. The four knots in the first segment are all set equal to the lowest data value of x and the other four additional knots are all set equal to the highest value: there is experimental evidence that coincident end-knots are best for numerical accuracy. The complete array must be left undisturbed if nag_2d_spline_eval (e02dec) or nag_2d_spline_eval_rect (e02dfc) is to be used subsequently.

ny – Integer *Input*

On entry: **ny** must specify the total number of knots associated with the variable y.

It is such that $\mathbf{ny} - 8$ is the number of interior knots.

Constraint: $\mathbf{ny} \geq 8$.

mu – double *Input/Output*

On entry: $\mathbf{mu}[i + 4]$ must contain the ith interior knot μ_{i+4} associated with the variable y, $i = 1, 2, \ldots, \mathbf{ny} - 8$.

On exit: the same remarks apply to **mu** as to **lamda** above, with **y** replacing **x**, and y replacing x.

c – double *Output*

On exit: gives the coefficients of the fit. $\mathbf{c}((\mathbf{ny} - 4) \times (i - 1) + j)$ is the coefficient c_{ij} of Sections 3 and 9, for $i = 1, 2, \ldots \mathbf{nx} - 4$ and $j = 1, 2, \ldots \mathbf{ny} - 4$. These coefficients are used by nag_2d_spline_eval (e02dec) or nag_2d_spline_eval_rect (e02dfc) to calculate values of the fitted function.

In normal usage, the call to nag_2d_spline_fit_panel (e02dac) follows a call to nag_2d_spline_interpolant (e01dac), nag_2d_spline_fit_grid (e02dcc) or nag_2d_spline_fit_scat

(e02ddc), in which case, members of the structure **spline** will have been set up correctly for input to nag_2d_spline_fit_panel (e02dac).

12: **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

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_CONSTRAINT

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

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

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: **m** > 1.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_KNOTS_COINCIDE

More than four knots coincide at a single point.

NE_KNOTS_CONS

At least one set of knots is not in nondecreasing order.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_PANEL_ORDER

Array **point** does not indicate the data points in panel order.

NE_WEIGHT_ZERO

All the weights are zero, or rank determined as zero.

7 Accuracy

The computation of the B-splines and reduction of the observation matrix to triangular form are both numerically stable.

8 Parallelism and Performance

nag_2d_spline_fit_panel (e02dac) is not threaded in any implementation.

9 Further Comments

The time taken is approximately proportional to the number of data points, m, and to $(3 \times (\mathbf{spline.ny} - 4) + 4)^2$.

The B-spline representation of the bicubic spline is

$$s(x,y) = \sum_{i,j} c_{ij} M_i(x) N_j(y)$$

summed over $i = 1, 2, \ldots, \mathbf{spline.nx} - 4$ and over $j = 1, 2, \ldots, \mathbf{spline.ny} - 4$. Here $M_i(x)$ and $N_j(y)$ denote normalized cubic B-splines, the former defined on the knots $\lambda_i, \lambda_{i+1}, \ldots, \lambda_{i+4}$ and the latter on the knots $\mu_j, \mu_{j+1}, \ldots, \mu_{j+4}$. For further details, see Hayes and Halliday (1974) for bicubic splines and de Boor (1972) for normalized B-splines.

The choice of the interior knots, which help to determine the spline's shape, must largely be a matter of trial and error. It is usually best to start with a small number of knots and, examining the fit at each stage, add a few knots at a time in places where the fit is particularly poor. In intervals of x or y where the surface represented by the data changes rapidly, in function value or derivatives, more knots will be needed than elsewhere. In some cases guidance can be obtained by analogy with the case of coincident knots: for example, just as three coincident knots can produce a discontinuity in slope, three close knots can produce rapid change in slope. Of course, such rapid changes in behaviour must be adequately represented by the data points, as indeed must the behaviour of the surface generally, if a satisfactory fit is to be achieved. When there is no rapid change in behaviour, equally-spaced knots will often suffice.

In all cases the fit should be examined graphically before it is accepted as satisfactory.

The fit obtained is not defined outside the rectangle

$$\lambda_4 \leq x \leq \lambda_{\mathbf{spline.nx}-3}, \mu_4 \leq y \leq \mu_{\mathbf{spline.ny}-3}.$$

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 $a \leq x \leq b$, $c \leq y \leq d$ defines the enlarged rectangle. In the case when the data are adequate to make the least squares solution unique ($\mathbf{rank} = (\mathbf{spline.nx} - 4) \times (\mathbf{spline.ny} - 4)$), this enlargement will not affect the fit over the original rectangle, except for possibly enlarged rounding errors, and will simply continue the bicubic polynomials in the panels bordering the rectangle out to the new boundaries: in other cases the fit will be affected. Even using the original rectangle there may be regions within it, particularly at its corners, which lie outside the data region and where, therefore, the fit will be unreliable. For example, if there is no data point in panel 1 of Figure 1 in Section 5, the least squares criterion leaves the spline indeterminate in this panel: the minimal spline determined by the function in this case passes through the value zero at the point (λ_4, μ_4).

10 Example

This example reads a value for ϵ, and a set of data points, weights and knot positions. If there are more y knots than x knots, it interchanges the x and y axes. It calls nag_2d_panel_sort (e02zac) to sort the data points into panel order, nag_2d_spline_fit_panel (e02dac) to fit a bicubic spline to them, and nag_2d_spline_eval (e02dec) to evaluate the spline at the data points.

Finally it prints:

– the weighted sum of squares of residuals computed from the linear equations;

– the rank determined by nag_2d_spline_fit_panel (e02dac);

– data points, fitted values and residuals in panel order;

– the weighted sum of squares of the residuals; and

– the coefficients of the spline fit.

The program is written to handle any number of datasets.

Note: the data supplied in this example is **not typical** of a realistic problem: the number of data points would normally be much larger (in which case the array dimensions would have to be increased); and the value of ϵ would normally be much smaller on most machines (see Section 5; the relatively large value of 10^{-6} has been chosen in order to illustrate a minimal least squares solution when **rank** $<$ (**spline**.**nx** $-$ 4) $\times$ (**spline**.**ny** $-$ 4); in this example (**spline**.**nx** $-$ 4) $\times$ (**spline**.**ny** $-$ 4) $=$ 24).

10.1 Program Text

```
/* nag_2d_spline_fit_panel  (e02dac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

int main(void)
{
  /* Initialized data */
  char label[] = "xy";

  /* Scalars */
  double d, eps, sigma, sum, temp;
  Integer exit_status = 0, i, iadres, itemp, j, m, nc, np, npoint, px, py,
          rank;

  /* Arrays */
  double *dl = 0, *f = 0, *ff = 0, *lamda = 0, *mu = 0, *w = 0, *x = 0;
  double *y = 0;
  Integer *point = 0;

  /* Nag Types */
  Nag_2dSpline spline;
  NagError fail;

  exit_status = 0;
  INIT_FAIL(fail);

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

  printf("nag_2d_spline_fit_panel (e02dac) Example Program Results\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
```

```
#ifdef _WIN32
  while (scanf_s("%lf", &eps) != EOF && exit_status == 0)
#else
  while (scanf("%lf", &eps) != EOF && exit_status == 0)
#endif
  {
    /* Read data, interchanging X and Y axes if PX.LT.PY */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%*[^\n] ", &m);
#else
    scanf("%" NAG_IFMT "%*[^\n] ", &m);
#endif
    if (m > 1) {
      /* Allocate memory */
      if (!(f = NAG_ALLOC(m, double)) ||
          !(ff = NAG_ALLOC(m, double)) ||
          !(w = 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;
      goto END;
    }
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &px, &py);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &px, &py);
#endif
    if (px < 8 && py < 8) {
      printf("px or py is too small.\n");
      exit_status = 1;
      goto END;
    }
    nc = (px - 4) * (py - 4);
    np = (px - 7) * (py - 7);
    npoint = m + (px - 7) * (py - 7);

    /* Allocate memory */
    if (!(dl = NAG_ALLOC(nc, double)) ||
        !(point = NAG_ALLOC(npoint, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

    if (px < py) {
      itemp = px;
      px = py;
      py = itemp;
      itemp = 1;
      /* Allocate memory */
      if (!(lamda = NAG_ALLOC(px, double)) || !(mu = NAG_ALLOC(py, double))
            )
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }

      for (i = 0; i < m; ++i)
#ifdef _WIN32
        scanf_s("%lf%lf%lf%lf", &y[i], &x[i], &f[i], &w[i]);
#else
```

```
        scanf("%lf%lf%lf%lf", &y[i], &x[i], &f[i], &w[i]);
#endif
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif

      if (py > 8) {
#ifdef _WIN32
        for (j = 4; j < py - 4; ++j)
          scanf_s("%lf", &mu[j]);
#else
        for (j = 4; j < py - 4; ++j)
          scanf("%lf", &mu[j]);
#endif
#ifdef _WIN32
        scanf_s("%*[^\n] ");
#else
        scanf("%*[^\n] ");
#endif
      }
      if (px > 8) {
#ifdef _WIN32
        for (j = 4; j < px - 4; ++j)
          scanf_s("%lf", &lamda[j]);
#else
        for (j = 4; j < px - 4; ++j)
          scanf("%lf", &lamda[j]);
#endif
#ifdef _WIN32
        scanf_s("%*[^\n] ");
#else
        scanf("%*[^\n] ");
#endif
      }
    }
    else {
      /* Allocate memory */
      if (!(lamda = NAG_ALLOC(px, double)) || !(mu = NAG_ALLOC(py, double))
            )
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }
      itemp = 0;
      for (i = 0; i < m; ++i)
#ifdef _WIN32
        scanf_s("%lf%lf%lf%lf", &x[i], &y[i], &f[i], &w[i]);
#else
        scanf("%lf%lf%lf%lf", &x[i], &y[i], &f[i], &w[i]);
#endif
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif

      if (px > 8) {
#ifdef _WIN32
        for (j = 4; j < px - 4; ++j)
          scanf_s("%lf", &lamda[j]);
#else
        for (j = 4; j < px - 4; ++j)
          scanf("%lf", &lamda[j]);
#endif
#ifdef _WIN32
        scanf_s("%*[^\n] ");
#else
        scanf("%*[^\n] ");
```

```
#endif
      }
      if (py > 8) {
#ifdef _WIN32
        for (j = 4; j < py - 4; ++j)
          scanf_s("%lf", &mu[j]);
#else
        for (j = 4; j < py - 4; ++j)
          scanf("%lf", &mu[j]);
#endif
#ifdef _WIN32
        scanf_s("%*[^\n] ");
#else
        scanf("%*[^\n] ");
#endif
      }
    }

    printf("\nInterior %1.1s -knots\n", label + itemp);
    for (j = 4; j < px - 4; ++j)
      printf("%11.4f\n", lamda[j]);
    if (px == 8)
      printf("None\n");

    printf("\nInterior %1.1s -knots\n", label + (2 - itemp - 1));
    for (j = 4; j < py - 4; ++j)
      printf("%1s%11.4f\n", "", mu[j]);
    if (py == 8)
      printf("None\n");

    /* nag_2d_panel_sort (e02zac).
     * Sort two-dimensional data into panels for fitting bicubic
     * splines
     */
    nag_2d_panel_sort(px, py, lamda, mu, m, x, y, point, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_2d_panel_sort (e02zac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    /* Fit bicubic spline to data points */
    spline.nx = px;
    spline.ny = py;

    if (!(spline.c = NAG_ALLOC((spline.nx - 4) * (spline.ny - 4), double)) ||
        !(spline.lamda = NAG_ALLOC(spline.nx, double)) ||
        !(spline.mu = NAG_ALLOC(spline.ny, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

    for (i = 0; i < spline.nx; i++)
      spline.lamda[i] = lamda[i];
    for (i = 0; i < spline.ny; i++)
      spline.mu[i] = mu[i];

    /* nag_2d_spline_fit_panel (e02dac).
     * Least squares surface fit, bicubic splines
     */
    nag_2d_spline_fit_panel(m, x, y, f, w, point, dl, eps, &sigma, &rank,
                            &spline, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_2d_spline_fit_panel (e02dac).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
```

```
    printf("\nSum of squares of residual RHS%16.3e\n", sigma);
    printf("\nRank%5" NAG_IFMT "\n", rank);

    /* nag_2d_spline_eval (e02dec).
     * Evaluation of bicubic spline, at a set of points
     */
    nag_2d_spline_eval(m, x, y, ff, &spline, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_2d_spline_eval (e02dec).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    sum = 0.;
    if (itemp == 1)
      printf("\nx and y have been interchanged\n\n");

    /*Output data points, fitted values and residuals */
    printf("      X          Y          Data       Fit      Residual\n");
    for (i = 0; i < np; ++i) {
      iadres = i + m;
      while ((iadres = point[iadres] - 1) >= 0) {
        temp = ff[iadres] - f[iadres];

        printf("%11.4f%11.4f%11.4f%11.4f%12.3e\n", x[iadres],
               y[iadres], f[iadres], ff[iadres], temp);
        /* Computing 2nd power */
        d = temp * w[iadres];
        sum += d * d;
      }
    }

    printf("\nSum of squared residuals%16.3e\n", sum);
    printf("\nSpline coefficients\n");
    for (i = 0; i < px - 4; ++i) {
      for (j = 0; j < py - 4; ++j)
        printf("%11.4f", spline.c[i * (py - 4) + j]);
      printf("\n");
    }
  END:
    NAG_FREE(dl);
    NAG_FREE(f);
    NAG_FREE(ff);
    NAG_FREE(lamda);
    NAG_FREE(mu);
    NAG_FREE(w);
    NAG_FREE(x);
    NAG_FREE(y);
    NAG_FREE(point);
    NAG_FREE(spline.lamda);
    NAG_FREE(spline.mu);
    NAG_FREE(spline.c);
  }
  return exit_status;
}
```

10.2 Program Data

```
nag_2d_spline_fit_panel (e02dac) Example Program Data
0.000001
  30
   8
  10
   -0.52    0.60    0.93   10.
   -0.61   -0.95   -1.79   10.
    0.93    0.87    0.36   10.
    0.09    0.84    0.52   10.
    0.88    0.17    0.49   10.
   -0.70   -0.87   -1.76   10.
    1.00    1.00    0.33    1.
```

```
    1.00    0.10    0.48    1.
    0.30    0.24    0.65    1.
   -0.77   -0.77   -1.82    1.
   -0.23    0.32    0.92    1.
   -1.00    1.00    1.00    1.
   -0.26   -0.63    8.88    1.
   -0.83   -0.66   -2.01    1.
    0.22    0.93    0.47    1.
    0.89    0.15    0.49    1.
   -0.80    0.99    0.84    1.
   -0.88   -0.54   -2.42    1.
    0.68    0.44    0.47    1.
   -0.14   -0.72    7.15    1.
    0.67    0.63    0.44    1.
   -0.90   -0.40   -3.34    1.
   -0.84    0.20    2.78    1.
    0.84    0.43    0.44    1.
    0.15    0.28    0.70    1.
   -0.91   -0.24   -6.52    1.
   -0.35    0.86    0.66    1.
   -0.16   -0.41    2.32    1.
   -0.35   -0.05    1.66    1.
   -1.00   -1.00   -1.00    1.
  -0.5
   0.0
```

10.3 Program Results

```
nag_2d_spline_fit_panel (e02dac) Example Program Results

Interior y -knots
    -0.5000
     0.0000

Interior x -knots
None

Sum of squares of residual RHS       1.467e+01

Rank   22

x and y have been interchanged

      X          Y          Data       Fit       Residual
    -0.9500    -0.6100    -1.7900    -1.7931  -3.126e-03
    -0.8700    -0.7000    -1.7600    -1.7521   7.893e-03
    -0.7700    -0.7700    -1.8200    -2.4301  -6.101e-01
    -0.6300    -0.2600     8.8800     7.6346  -1.245e+00
    -0.6600    -0.8300    -2.0100    -1.5815   4.285e-01
    -0.5400    -0.8800    -2.4200    -2.6795  -2.595e-01
    -0.7200    -0.1400     7.1500     7.5708   4.208e-01
    -1.0000    -1.0000    -1.0000    -1.0228  -2.277e-02
    -0.4000    -0.9000    -3.3400    -4.6955  -1.356e+00
    -0.2400    -0.9100    -6.5200    -4.7072   1.813e+00
    -0.4100    -0.1600     2.3200     2.7039   3.839e-01
    -0.0500    -0.3500     1.6600     2.2865   6.265e-01
     0.6000    -0.5200     0.9300     0.9441   1.407e-02
     0.8700     0.9300     0.3600     0.3529  -7.097e-03
     0.8400     0.0900     0.5200     0.5024  -1.761e-02
     0.1700     0.8800     0.4900     0.4705  -1.951e-02
     1.0000     1.0000     0.3300     0.6315   3.015e-01
     0.1000     1.0000     0.4800     1.4910   1.011e+00
     0.2400     0.3000     0.6500     0.9241   2.741e-01
     0.3200    -0.2300     0.9200    -0.3692  -1.289e+00
     1.0000    -1.0000     1.0000     1.0835   8.352e-02
     0.9300     0.2200     0.4700     1.4912   1.021e+00
     0.1500     0.8900     0.4900     0.4414  -4.861e-02
     0.9900    -0.8000     0.8400     0.5495  -2.905e-01
     0.4400     0.6800     0.4700     1.5862   1.116e+00
     0.6300     0.6700     0.4400     0.6288   1.888e-01
```

```
     0.2000    -0.8400     2.7800     1.7123  -1.068e+00
     0.4300     0.8400     0.4400     0.6888   2.488e-01
     0.2800     0.1500     0.7000     0.7713   7.134e-02
     0.8600    -0.3500     0.6600     0.9347   2.747e-01

Sum of squared residuals       1.467e+01

Spline coefficients
    -1.0228   115.4668  -433.5558   -68.1973
    24.8426  -140.1485   258.5042    15.6756
   -29.4878   132.2933  -173.5103    20.0983
     9.9575   -51.6200    67.6666    -5.8765
    10.0577     4.7543   -15.3533    -0.3260
     1.0835    -2.7932     7.7708     0.6315
```

Example Program
Evaluation of Least-squares Bi-cubic Spline Fit