

# NAG Library Function Document

## nag\_2d\_spline\_interpolant (e01dac)

### 1 Purpose

nag\_2d\_spline\_interpolant (e01dac) computes a bicubic spline interpolating surface through a set of data values, given on a rectangular grid in the  $x$ - $y$  plane.

### 2 Specification

```
#include <nag.h>
#include <nage01.h>

void nag_2d_spline_interpolant (Integer mx, Integer my, const double x[],
    const double y[], const double f[], Nag_2dSpline *spline,
    NagError *fail)
```

### 3 Description

nag\_2d\_spline\_interpolant (e01dac) determines a bicubic spline interpolant to the set of data points  $(x_q, y_r, f_{q,r})$ , for  $q = 1, 2, \dots, m_x$  and  $r = 1, 2, \dots, m_y$ . The spline is given in the B-spline representation

$$s(x, y) = \sum_{i=1}^{m_x} \sum_{j=1}^{m_y} c_{ij} M_i(x) N_j(y)$$

such that

$$s(x_q, y_r) = f_{q,r},$$

where  $M_i(x)$  and  $N_j(y)$  denote normalized cubic B-splines, the former defined on the knots  $\lambda_i$  to  $\lambda_{i+4}$  and the latter on the knots  $\mu_j$  to  $\mu_{j+4}$ , and the  $c_{ij}$  are the spline coefficients. These knots, as well as the coefficients, are determined by the function, which is derived from the routine B2IRE in Anthony *et al.* (1982). The method used is described in Section 9.1.

For further information on splines, see Hayes and Halliday (1974) for bicubic splines and de Boor (1972) for normalized B-splines.

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 9.2.

### 4 References

Anthony G T, Cox M G and Hayes J G (1982) *DASL – Data Approximation Subroutine Library* National Physical Laboratory

Cox M G (1975) An algorithm for spline interpolation *J. Inst. Math. Appl.* **15** 95–108

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

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: **mx** – Integer *Input*  
 2: **my** – Integer *Input*

*On entry:* **mx** and **my** must specify  $m_x$  and  $m_y$  respectively, the number of points along the  $x$  and  $y$  axis that define the rectangular grid.

*Constraint:* **mx**  $\geq 4$  and **my**  $\geq 4$ .

- 3: **x[**mx**]** – const double *Input*  
 4: **y[**my**]** – const double *Input*

*On entry:* **x**[ $q - 1$ ] and **y**[ $r - 1$ ] must contain  $x_q$ , for  $q = 1, 2, \dots, m_x$ , and  $y_r$ , for  $r = 1, 2, \dots, m_y$ , respectively.

*Constraints:*

$$\begin{aligned} \mathbf{x}[q - 1] &< \mathbf{x}[q], \text{ for } q = 1, 2, \dots, m_x - 1; \\ \mathbf{y}[r - 1] &< \mathbf{y}[r], \text{ for } r = 1, 2, \dots, m_y - 1. \end{aligned}$$

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

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

- 6: **spline** – Nag\_2dSpline \*

Pointer to structure of type Nag\_2dSpline with the following members:

- nx** – Integer *Output*  
**ny** – Integer *Output*

*On exit:* **nx** and **ny** contain  $m_x + 4$  and  $m_y + 4$ , the total number of knots of the computed spline with respect to the  $x$  and  $y$  variables, respectively.

- lamda** – double \* *Output*

*On exit:* the pointer to which memory of size **nx** is internally allocated. **lamda** contains the complete set of knots  $\lambda_i$  associated with the  $x$  variable, i.e., the interior knots **lamda**[4], **lamda**[5], ..., **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.

- mu** – double \* *Output*

*On exit:* the pointer to which memory of size **ny** is internally allocated. **mu** contains the corresponding complete set of knots  $\mu_i$  associated with the  $y$  variable.

- c** – double \* *Output*

*On exit:* the pointer to which memory of size **mx**  $\times$  **my** is internally allocated. **c** holds the coefficients of the spline interpolant. **c**[ $m_y \times (i - 1) + j - 1$ ] contains the coefficient  $c_{ij}$  described 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\_interpolant (e01dac) with the same **spline**, you should free these pointers using the NAG macro NAG\_FREE. This storage will not have been allocated if this function returns with **fail.code**  $\neq$  NE\_NOERROR.

- 7: **fail** – NagError \* *Input/Output*

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

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

### NE\_DATA\_ILL\_CONDITIONED

An intermediate set of linear equations is singular, the data is too ill-conditioned to compute B-spline coefficients.

### NE\_INT\_ARG\_LT

On entry,  $\mathbf{mx} = \langle \text{value} \rangle$ .

Constraint:  $\mathbf{mx} \geq 4$ .

On entry,  $\mathbf{my} = \langle \text{value} \rangle$ .

Constraint:  $\mathbf{my} \geq 4$ .

### NE\_NOT\_STRICTLY\_INCREASING

The sequence  $\mathbf{x}$  is not strictly increasing:  $\mathbf{x}[\langle \text{value} \rangle] = \langle \text{value} \rangle$ ,  $\mathbf{x}[\langle \text{value} \rangle] = \langle \text{value} \rangle$ .

The sequence  $\mathbf{y}$  is not strictly increasing:  $\mathbf{y}[\langle \text{value} \rangle] = \langle \text{value} \rangle$ ,  $\mathbf{y}[\langle \text{value} \rangle] = \langle \text{value} \rangle$ .

## 7 Accuracy

The main sources of rounding errors are in steps 1, 3, 6 and 7 of the algorithm described in Section 9.1. It can be shown (Cox (1975)) that the matrix  $A_x$  formed in step 2 has elements differing relatively from their true values by at most a small multiple of  $3\epsilon$ , where  $\epsilon$  is the *machine precision*.  $A_x$  is ‘totally positive’, and a linear system with such a coefficient matrix can be solved quite safely by elimination without pivoting. Similar comments apply to steps 6 and 7. Thus the complete process is numerically stable.

## 8 Parallelism and Performance

Not applicable.

## 9 Further Comments

The time taken by `nag_2d_spline_interpolant` (e01dac) is approximately proportional to  $m_x m_y$ .

### 9.1 Outline of Method Used

The process of computing the spline consists of the following steps:

1. choice of the interior  $x$ -knots  $\lambda_5, \lambda_6, \dots, \lambda_{m_x}$  as  $\lambda_i = x_{i-2}$ , for  $i = 5, 6, \dots, m_x$ ,
2. formation of the system

$$A_x E = F,$$

where  $A_x$  is a band matrix of order  $m_x$  and bandwidth 4, containing in its  $q$ th row the values at  $x_q$  of the B-splines in  $x$ ,  $F$  is the  $m_x$  by  $m_y$  rectangular matrix of values  $f_{q,r}$ , and  $E$  denotes an  $m_x$  by  $m_y$  rectangular matrix of intermediate coefficients,

3. use of Gaussian elimination to reduce this system to band triangular form,
4. solution of this triangular system for  $E$ ,
5. choice of the interior  $y$  knots  $\mu_5, \mu_6, \dots, \mu_{m_y}$  as  $\mu_i = y_{i-2}$ , for  $i = 5, 6, \dots, m_y$ ,
6. formation of the system

$$A_y C^T = E^T,$$

where  $A_y$  is the counterpart of  $A_x$  for the  $y$  variable, and  $C$  denotes the  $m_x$  by  $m_y$  rectangular matrix of values of  $c_{ij}$ ,

7. use of Gaussian elimination to reduce this system to band triangular form,
8. solution of this triangular system for  $C^T$  and hence  $C$ .

For computational convenience, steps 2 and 3, and likewise steps 6 and 7, are combined so that the formation of  $A_x$  and  $A_y$  and the reductions to triangular form are carried out one row at a time.

## 9.2 Evaluation of Computed Spline

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

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

where **spline** is a structure of type Nag\_2dSpline which is the output argument of nag\_2d\_spline\_interpolant (e01dac).

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  $q = 1, 2, \dots, \mathbf{kx}$ , and the  $y$  coordinates stored in  $\mathbf{ty}[r-1]$ , for  $r = 1, 2, \dots, \mathbf{ky}$ , 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 the output argument of nag\_2d\_spline\_interpolant (e01dac). 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 program reads in values of  $m_x$ ,  $x_q$ , for  $q = 1, 2, \dots, m_x$ ,  $m_y$  and  $y_r$ , for  $r = 1, 2, \dots, m_y$ , followed by values of the ordinates  $f_{q,r}$  defined at the grid points  $(x_q, y_r)$ . It then calls nag\_2d\_spline\_interpolant (e01dac) to compute a bicubic spline interpolant of the data values, and prints the values of the 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_interpolant (e01dac) 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 <nage01.h>
#include <nage02.h>

#define F(I, J) f[my*(I)+(J)]
#define FG(I, J) fg[npy*(I)+(J)]
#define C(I, J) spline.c[my*(I)+(J)]
int main(void)
{
    Integer      exit_status = 0, i, j, mx, my, npx, npy;
    NagError     fail;
    Nag_2dSpline spline;
    double       *f = 0, *fg = 0, step, *tx = 0, *ty = 0, *x = 0, xhi, xlo;
    double       *y = 0, yhi, ylo;
```

```

INIT_FAIL(fail);

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

printf(
    "nag_2d_spline_interpolant (e01dac) Example Program Results\n");
#ifdef _WIN32
    scanf_s("%*[\n]"); /* Skip heading in data file */
#else
    scanf("%*[\n]"); /* Skip heading in data file */
#endif
/* Read the number of x points, mx, and the values of the
 * x co-ordinates.
 */
#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;
        }
    for (i = 0; i < mx; i++)
#ifdef _WIN32
        scanf_s("%lf", &x[i]);
#else
        scanf("%lf", &x[i]);
#endif
/* Read the number of y points, my, and the values of the
 * y co-ordinates.
 */
    for (i = 0; i < my; i++)
#ifdef _WIN32
        scanf_s("%lf", &y[i]);
#else
        scanf("%lf", &y[i]);
#endif
/* Read the function values at the grid points. */
    for (j = 0; j < my; j++)
        for (i = 0; i < mx; i++)
#ifdef _WIN32
            scanf_s("%lf", &F(i, j));
#else
            scanf("%lf", &F(i, j));
#endif
/* Generate the (x,y,f) interpolating bicubic B-spline. */
/* nag_2d_spline_interpolant (e01dac).
 * Interpolating function, bicubic spline interpolant, two
 * variables
 */
nag_2d_spline_interpolant(mx, my, x, y, f, &spline, &fail);
if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_2d_spline_interpolant (e01dac).\n%s\n",

```

```

        fail.message);
    exit_status = 1;
    goto END;
}

/* 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":" ");
/* Print the spline coefficients. */
printf("\nThe B-Spline coefficients:\n");
for (i = 0; i < mx; i++)
{
    for (j = 0; j < my; j++)
        printf("%9.4f", C(i, j));
    printf("\n");
}

/* Evaluate the spline on a regular rectangular grid at npix*ncpy
 * points over the domain (xlo to xhi) x (ylo to yhi).
 */
#ifdef _WIN32
    scanf_s("%NAG_IFMT%lf%lf", &npix, &xlo, &xhi);
#else
    scanf("%NAG_IFMT%lf%lf", &npix, &xlo, &xhi);
#endif
#ifdef _WIN32
    scanf_s("%NAG_IFMT%lf%lf", &ncpy, &ylo, &yhi);
#else
    scanf("%NAG_IFMT%lf%lf", &ncpy, &ylo, &yhi);
#endif
if (npix >= 1 && ncpy >= 1)
{
    if (!(fg = NAG_ALLOC(npix*ncpy, double)) ||
        !(tx = NAG_ALLOC(npix, double)) ||
        !(ty = NAG_ALLOC(ncpy, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else
{
    printf("Invalid npix or ncpy.\n");
    exit_status = 1;
    return exit_status;
}
step = (xhi-xlo)/(double)(npix-1);
printf("\nSpline evaluated on a regular mesh "
       " (x across, y down): \n ");
/* Generate nx equispaced x co-ordinates. */
for (i = 0; i < npix; i++)
{
    tx[i] = MIN(xlo+i*step, xhi);
    printf(" %5.2f ", tx[i]);
}
step = (yhi-ylo)/(double)(ncpy-1);
for (i = 0; i < ncpy; i++)
    ty[i] = MIN(ylo+i*step, yhi);

/* Evaluate the spline. */
/* nag_2d_spline_eval_rect (e02dfc).
 * Evaluation of bicubic spline, at a mesh of points
 */
nag_2d_spline_eval_rect(npix, ncpy, tx, ty, 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;
}

/* Print the results. */
printf("\n");
for (j = 0; j < npy; j++)
{
    printf("%5.2f", ty[j]);
    for (i = 0; i < npx; i++)
        printf("%8.3f ", FG(i, j));
    printf("\n");
}
/* Free memory allocated by nag_2d_spline_interpolant (e01dac) */
END:
NAG_FREE(spline.lamda);
NAG_FREE(spline.mu);
NAG_FREE(spline.c);
NAG_FREE(f);
NAG_FREE(x);
NAG_FREE(y);
NAG_FREE(fg);
NAG_FREE(tx);
NAG_FREE(ty);
return exit_status;
}

```

## 10.2 Program Data

nag\_2d\_spline\_interpolant (e01dac) Example Program Data

```

7 6
1.00  1.10  1.30  1.50  1.60  1.80  2.00
0.00  0.10  0.40  0.70  0.90  1.00
1.00  1.21  1.69  2.25  2.56  3.24  4.00
1.10  1.31  1.79  2.35  2.66  3.34  4.10
1.40  1.61  2.09  2.65  2.96  3.64  4.40
1.70  1.91  2.39  2.95  3.26  3.94  4.70
1.90  2.11  2.59  3.15  3.46  4.14  4.90
2.00  2.21  2.69  3.25  3.56  4.24  5.00
6  1.0  2.0
6  0.0  1.0

```

## 10.3 Program Results

nag\_2d\_spline\_interpolant (e01dac) Example Program Results

Distinct knots in x direction located at

1.0000	1.3000	1.5000	1.6000	2.0000
--------	--------	--------	--------	--------

Distinct knots in y direction located at

0.0000	0.4000	0.7000	1.0000
--------	--------	--------	--------

The B-Spline coefficients:

1.0000	1.1333	1.3667	1.7000	1.9000	2.0000
1.2000	1.3333	1.5667	1.9000	2.1000	2.2000
1.5833	1.7167	1.9500	2.2833	2.4833	2.5833
2.1433	2.2767	2.5100	2.8433	3.0433	3.1433
2.8667	3.0000	3.2333	3.5667	3.7667	3.8667
3.4667	3.6000	3.8333	4.1667	4.3667	4.4667
4.0000	4.1333	4.3667	4.7000	4.9000	5.0000

Spline evaluated on a regular mesh (x across, y down):

	1.00	1.20	1.40	1.60	1.80	2.00
0.00	1.000	1.440	1.960	2.560	3.240	4.000

0.20	1.200	1.640	2.160	2.760	3.440	4.200
0.40	1.400	1.840	2.360	2.960	3.640	4.400
0.60	1.600	2.040	2.560	3.160	3.840	4.600
0.80	1.800	2.240	2.760	3.360	4.040	4.800
1.00	2.000	2.440	2.960	3.560	4.240	5.000

---