# nag_2d_spline_eval_rect (e02dfc)

## 1. Purpose

**nag_2d_spline_eval_rect (e02dfc)** calculates values of a bicubic spline from its B-spline representation. The spline is evaluated at all points on a rectangular grid.

## 2. Specification

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

void nag_2d_spline_eval_rect(Integer mx, Integer my, double x[], double y[],
            double ff[], Nag_2dSpline *spline, NagError *fail)
```

## 3. Description

This function calculates values of the bicubic spline $s(x,y)$ on a rectangular grid of points in the $x$-$y$ plane, from its augmented knot sets $\{\lambda\}$ and $\{\mu\}$ and from the coefficients $c_{ij}$, for $i = 1,2,\ldots,$**spline.nx**$-4$; $j = 1,2,\ldots,$**spline.ny**$-4$, in its B-spline representation

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

Here $M_i(x)$ and $N_j(y)$ denote normalised cubic B-splines, the former defined on the knots $\lambda_i$ to $\lambda_{i+4}$ and the latter on the knots $\mu_j$ to $\mu_{j+4}$.

The points in the grid are defined by co-ordinates $x_q$, for $q = 1,2,\ldots,m_x$, along the $x$ axis, and co-ordinates $y_r$, for $r = 1,2,\ldots,m_y$ along the $y$ axis.

This function may be used to calculate values of a bicubic spline given in the form produced by nag_2d_spline_interpolant (e01dac), nag_2d_spline_fit_grid (e02dcc) and nag_2d_spline_fit_scat (e02ddc). It is derived from the routine B2VRE in Anthony *et al* (1982).

## 4. Parameters

**mx**
**my**

Input: **mx** and **my** must specify $m_x$ and $m_y$ respectively, the number of points along the $x$ and $y$ axes that define the rectangular grid.
Constraint: **mx** $\geq 1$ and **my** $\geq 1$.

**x[mx]**
**y[my]**

Input: **x** and **y** must contain $x_q$, for $q = 1,2,\ldots,m_x$, and $y_r$, for $r = 1,2,\ldots,m_y$, respectively. These are the $x$ and $y$ co-ordinates that define the rectangular grid of points at which values of the spline are required.
Constraints: **x** and **y** must satisfy
**spline.lamda**[3] $\leq$ **x**[$q-1$] $<$ **x**[q] $\leq$ **spline.lamda**[**spline.nx**$-4$], for $q = 1,2,\ldots,m_x-1$, and
**spline.mu**[3] $\leq$ **y**[$r-1$] $<$ **y**[r] $\leq$ **spline.mu**[**spline.ny**$-4$], for $r = 1,2,\ldots,m_y-1$.
The spline representation is not valid outside these intervals.

**ff[mx*my]**

Output: **ff[my** $\times$ $(q-1)+r-1$] contains the value of the spline at the point $(x_q, y_r)$, for $q = 1,2,\ldots,m_x$; $r = 1,2,\ldots,m_y$.

**spline**

Input: Pointer to structure of type Nag_2dSpline with the following members:

**nx** - Integer

Input: **spline.nx** must specify the total number of knots associated with the variable $x$. It is such that **spline.nx**$-8$ is the number of interior knots.

Constraint: **spline.nx** $\geq 8$.

**lamda** - double *
Input: a pointer to which memory of size **spline.nx** must be allocated. **spline.lamda** must contain the complete sets of knots $\{\lambda\}$ associated with the $x$ variable.
Constraint: the knots must be in non-decreasing order, with **spline**.**lamda**[**spline**.**nx** $-4$] $>$ **spline**.**lamda**[3].

**ny** - Integer
Input: **spline.ny** must specify the total number of knots associated with the variable $y$. It is such that **spline**.**ny** $-8$ is the number of interior knots.
Constraint: **spline.ny** $\geq 8$.

**mu** - double *
Input: a pointer to which memory of size **spline.ny** must be allocated. **spline.mu** must contain the complete sets of knots $\{\mu\}$ associated with the $y$ variable.
Constraint: the knots must be in non-decreasing order, with **spline**.**mu**[**spline**.**ny** $-4$] $>$ **spline**.**mu**[3].

**c** - double *
Input: a pointer to which memory of size (**spline**.**nx** $-4$) $\times$ (**spline**.**ny** $-4$) must be allocated. **spline**.**c**[(**spline**.**ny** $-4$) $\times (i-1)+j-1$] must contain the coefficient $c_{ij}$ described in Section 3, for $i = 1, 2, \ldots,$ **spline**.**nx** $-4$; $j = 1, 2, \ldots,$ **spline**.**ny** $-4$.

In normal usage, the call to nag_2d_spline_eval_rect 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_eval_rect.

**fail**
The NAG error parameter, see the Essential Introduction to the NAG C Library.

## 5. Error Indications and Warnings

**NE_INT_ARG_LT**
On entry, **mx** must not be less than 1: **mx** = ⟨*value*⟩.
On entry, **my** must not be less than 1: **my** = ⟨*value*⟩.
On entry, **spline.nx** must not be less than 8: **spline.nx** = ⟨*value*⟩.
On entry, **spline.ny** must not be less than 8: **spline.ny** = ⟨*value*⟩.

**NE_ALLOC_FAIL**
Memory allocation failed.

**NE_END_KNOTS_CONS**
On entry, the end knots must satisfy ⟨*value*⟩, ⟨*value*⟩ = ⟨*value*⟩, ⟨*value*⟩ = ⟨*value*⟩.

**NE_NOT_INCREASING**
The sequence **spline.lamda** is not increasing: **spline.lamda**[⟨*value*⟩] = ⟨*value*⟩, **spline.lamda**[⟨*value*⟩] = ⟨*value*⟩.
The sequence **spline.mu** is not increasing: **spline.mu**[⟨*value*⟩] = ⟨*value*⟩, **spline.mu**[⟨*value*⟩] = ⟨*value*⟩.

**NE_KNOTS_COORD_CONS**
On entry, the end knots and coordinates must satisfy **spline.lamda**[3] $\leq$ **x**[0] and **x**[**mx**]$-1$] $\leq$ **spline.lamda**[**spline.nx**$-4$]. **spline.lamda**[3] = ⟨*value*⟩, **x**[0] = ⟨*value*⟩, **x**[⟨*value*⟩] = ⟨*value*⟩, **spline.lamda**[⟨*value*⟩] = ⟨*value*⟩.
On entry, the end knots and coordinates must satisfy **spline.mu**[3] $\leq$ **y**[0] and **y**[**my**$-1$] $\leq$ **spline.mu**[**spline.ny**$-4$]. **spline.mu**[3] = ⟨*value*⟩, **y**[0] = ⟨*value*⟩, **y**[⟨*value*⟩] = ⟨*value*⟩, **spline.mu**[⟨*value*⟩] = ⟨*value*⟩.

**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*⟩.

## 6. Further Comments

Computation time is approximately proportional to $m_x m_y + 4(m_x + m_y)$.

### 6.1. Accuracy

The method used to evaluate the B-splines is numerically stable, in the sense that each computed value of $s(x_r, y_r)$ can be regarded as the value that would have been obtained in exact arithmetic from slightly perturbed B-spline coefficients. See Cox (1978) for details.

### 6.2. References

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

Cox M G (1978) The Numerical Evaluation of a Spline from its B-spline Representation *J. Inst. Math. Appl.* **21** 135–143.

## 7. See Also

nag_2d_spline_interpolant (e01dac)
nag_2d_spline_fit_grid (e02dcc)
nag_2d_spline_fit_scat (e02ddc)
nag_2d_spline_eval (e02dec)

## 8. Example

This program reads in knot sets **spline.lamda**[0],...,**spline.lamda**[**spline.nx**−1]*and***spline.mu**[0],..., **spline.mu**[**spline.ny**−1], and a set of bicubic spline coefficients $c_{ij}$. Following these are values for $m_x$ and the $x$ co-ordinates $x_q$, for $q = 1, 2, \ldots, m_x$, and values for $m_y$ and the $y$ co-ordinates $y_r$, for $r = 1, 2, \ldots, m_y$, defining the grid of points on which the spline is to be evaluated.

### 8.1. Program Text

```
/* nag_2d_spline_eval_rect(e02dfc) Example Program
 *
 * Copyright 1991 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 */

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

#define MXMAX 20
#define MYMAX 20
#define FF(I,J)  ff[my*(I)+(J)]

main()
{
  Integer i, j, mx, my;
  double x[MXMAX], y[MYMAX], ff[MXMAX*MYMAX];
  Nag_2dSpline spline;

  Vprintf("e02dfc Example Program Results\n");
  /* Skip heading in data file */
  Vscanf("%*[^\n]");
  /* Read mx and my, the number of grid points in the x and y
   * directions respectively.
   */
  Vscanf("%ld%ld",&mx,&my);
  if (mx<=MXMAX && my<=MYMAX)
    {
      /* Read spline.nx and spline.ny, the number of knots
       * in the x and y directions.
       */
      Vscanf("%ld%ld",&(spline.nx),&(spline.ny));
      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);
      if (spline.c != (double *)0 && spline.lamda != (double *)0
```

```
          && spline.mu != (double *)0)
        {

          /* Read the knots spline.lamda[0]...spline.lamda[nx-1]
           * and spline.mu[0]...spline.mu[ny-1].
           */
          for (i=0; i<spline.nx;  i++)
            Vscanf("%lf",&(spline.lamda[i]));
          for (i=0; i<spline.ny;  i++)
            Vscanf("%lf",&(spline.mu[i]));
          /* Read spline.c, the bicubic spline coefficients. */
          for (i=0; i<(spline.nx-4)*(spline.ny-4); i++)
            Vscanf("%lf",&(spline.c[i]));
          /* Read the x and y co-ordinates defining the evaluation grid. */
          for (i=0; i<mx;  i++)
            Vscanf("%lf",&x[i]);
          for (i=0; i<my;  i++)
            Vscanf("%lf",&y[i]);
          /* Evaluate the spline at the mx by my points. */
          e02dfc(mx, my, x, y, ff, &spline, NAGERR_DEFAULT);
          /* Print the result array. */
          Vprintf("Spline evaluated on x-y grid (x across, y down):\n      ");
          for (i=0; i<mx;  i++)
            Vprintf("%2.1f      ",x[i]);
          Vprintf("\n");
          for (j=0; j<my; j++)
            {
              Vprintf("%2.1f",y[j]);
              for (i=0; i<mx; i++)
                Vprintf("%8.3f%s",FF(i,j), (i%7==6 || i==mx-1) ? "\n" : " ");
            }
          NAG_FREE(spline.c);
          NAG_FREE(spline.lamda);
          NAG_FREE(spline.mu);
          exit(EXIT_SUCCESS);
        }
      else
        {
          Vfprintf(stderr,"Storage allocation failed.\n");
          exit(EXIT_FAILURE);
        }
    }
  else
    {
      Vfprintf(stderr, "mx or my is out of range: mx = %5ld, my = %5ld\n",
               mx,my);
      exit(EXIT_FAILURE);
    }
}
```

### 8.2. Program Data

```
e02dfc Example Program Data
7  6
11  10
1.0  1.0  1.0  1.0  1.3  1.5  1.6  2.0  2.0  2.0  2.0
0.0  0.0  0.0  0.0  0.4  0.7  1.0  1.0  1.0  1.0
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
1.0  1.1  1.3  1.4  1.5  1.7  2.0
0.0  0.2  0.4  0.6  0.8  1.0
```

### 8.3. Program Results

```
e02dfc Example Program Results
Spline evaluated on x-y grid (x across, y down):
     1.0      1.1      1.3      1.4      1.5      1.7      2.0
0.0  1.000    1.210    1.690    1.960    2.250    2.890    4.000
0.2  1.200    1.410    1.890    2.160    2.450    3.090    4.200
0.4  1.400    1.610    2.090    2.360    2.650    3.290    4.400
0.6  1.600    1.810    2.290    2.560    2.850    3.490    4.600
0.8  1.800    2.010    2.490    2.760    3.050    3.690    4.800
1.0  2.000    2.210    2.690    2.960    3.250    3.890    5.000
```