nag_2d_spline_eval (e02dec)

1. Purpose

nag_2d_spline_eval (e02dec) calculates values of a bicubic spline from its B-spline representation.

2. Specification

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

void nag_2d_spline_eval(Integer m, double x[], double y[], double ff[],
            Nag_2dSpline *spline, NagError *fail)
```

3. Description

This function calculates values of the bicubic spline $s(x, y)$ at prescribed points (x_r, y_r), for $r = 1, 2, \ldots, m$, 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 λ_i to λ_{i+4} and the latter on the knots μ_j to μ_{j+4}.

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

m

Input: m, the number of points at which values of the spline are required.
Constraint: **m** ≥ 1.

x[m]
y[m]

Input: **x** and **y** must contain x_r and y_r, for $r = 1, 2, \ldots, m$, respectively. These are the co-ordinates of the points at which values of the spline are required. The order of the points is immaterial.
Constraint: **x** and **y** must satisfy
spline.lamda[3] $\leq$ **x**$[r-1] \leq$ **spline.lamda**[**spline.nx**-4]
and
spline.mu[3] $\leq$ **y**$[r-1] \leq$ **spline.mu**[**spline.ny**-4], for $r = 1, 2, \ldots, m$.
The spline representation is not valid outside these intervals.

ff[m]

Output: **ff**$[r-1]$ contains the value of the spline at the point (x_r, y_r), for $r = 1, 2, \ldots, m$.

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 variables x. It is such that **spline.nx**-8 is the number of interior knots.
Constraint: **spline.nx** ≥ 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** ≥ 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) × (**spline.ny** − 4) must be allocated. **spline.c**[(**spline.ny** − 4) × $(i-1)+j-1$] 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 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.

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, **m** must not be less than 1: **m** = ⟨*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_POINT_OUTSIDE_RECT

On entry, point (**x**[⟨*value*⟩] = ⟨*value*⟩, **y**[⟨*value*⟩] = ⟨*value*⟩) lies outside the rectangle bounded by **spline.lamda**[3] = ⟨*value*⟩, **spline.lamda**[⟨*value*⟩] = ⟨*value*⟩, **spline.mu**[3] = ⟨*value*⟩, **spline.mu**[⟨*value*⟩] = ⟨*value*⟩.

6. Further Comments

Computation time is approximately proportional to the number of points, m, at which the evaluation is required.

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_rect (e02dfc)

8. Example

This program reads in knot sets **spline.lamda**[0],. . . ,**spline.lamda**[**spline.nx**−1] and **spline.mu**[0],. . . ,**spline.mu**[**spline.ny** and a set of bicubic spline coefficients c_{ij}. Following these are a value for m and the co-ordinates (x_r, y_r), for $r = 1, 2, \ldots, m$, at which the spline is to be evaluated.

8.1. Program Text

```
/* nag_2d_spline_eval(e02dec) Example Program
 *
 * Copyright 1991 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 */

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

#define MMAX 20

main()
{
  Integer i, m;
  double x[MMAX], y[MMAX], ff[MMAX];
  Nag_2dSpline spline;

  Vprintf("e02dec Example Program Results\n");
  Vscanf("%*[^\n]"); /* Skip heading in data file */
  /* Read m, the number of spline evaluation points. */
  Vscanf("%ld",&m);
  if (m<=MMAX)
    {
      /* Read nx and 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 lamda[0] .. lamda[nx-1] and mu[0] .. 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 c, the bicubic spline coefficients. */
          for (i=0; i<(spline.nx-4)*(spline.ny-4);
               Vscanf("%lf",&(spline.c[i])), i++);
          /* Read the x and y co-ordinates of the evaluation points. */
          for (i=0; i<m; i++)
            Vscanf("%lf%lf",&x[i],&y[i]);
          /* Evaluate the spline at the m points. */
          e02dec(m, x, y, ff, &spline, NAGERR_DEFAULT);
          /* Print the results. */
          Vprintf("      i          x[i]       y[i]       ff[i]\n");
          for (i=0; i<m; i++)
            Vprintf("%7ld    %11.3f%11.3f%11.3f\n",i,x[i],y[i],ff[i]);
          NAG_FREE(spline.lamda);
          NAG_FREE(spline.mu);
```

```
          NAG_FREE(spline.c);
          exit(EXIT_SUCCESS);
        }
      else
        {
          Vfprintf(stderr,"Storage allocation failed.\n");
          exit(EXIT_FAILURE);
        }
    }
  else
    {
      Vfprintf(stderr, "m is out of range: m = %5ld\n",m);
      exit(EXIT_FAILURE);
    }
}
```

8.2. Program Data

```
e02dec Example Program Data
7
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  0.0
1.1  0.1
1.5  0.7
1.6  0.4
1.9  0.3
1.9  0.8
2.0  1.0
```

8.3. Program Results

```
e02dec Example Program Results
      i          x[i]       y[i]       ff[i]
      0          1.000      0.000      1.000
      1          1.100      0.100      1.310
      2          1.500      0.700      2.950
      3          1.600      0.400      2.960
      4          1.900      0.300      3.910
      5          1.900      0.800      4.410
      6          2.000      1.000      5.000
```