NAG Library Function Document

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, const double x[], const double y[],
     double ff[], Nag_2dSpline *spline, NagError *fail)
```

3 Description

nag_2d_spline_eval (e02dec) 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** $- 4$ and $j = 1, 2, \ldots,$ **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 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}.

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 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

5 Arguments

1: **m** – Integer *Input*

On entry: m, the number of points at which values of the spline are required.

Constraint: **m** ≥ 1.

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

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

On entry: **x** and **y** must contain x_r and y_r, for $r = 1, 2, \ldots, m$, respectively. These are the coordinates 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].
The spline representation is not valid outside these intervals, for $r = 1, 2, \ldots, m$.

4: **ff**[**m**] – double *Output*

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

5: **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 **nx** $-$ 8 is the number of interior knots.

Constraint: **nx** ≥ 8.

lamda – double * *Input*

On entry: a pointer to which memory of size **nx** must be allocated. **lamda** must contain the complete sets of knots $\{\lambda\}$ associated with the x variable.

Constraint: the knots must be in nondecreasing order, with **lamda**[**nx** $-$ 4] $>$ **lamda**[3].

ny – Integer *Input*

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

It is such that **ny** $-$ 8 is the number of interior knots.

Constraint: **ny** ≥ 8.

mu – double * *Input*

On entry: a pointer to which memory of size **ny** must be allocated. **mu** must contain the complete sets of knots $\{\mu\}$ associated with the y variable.

Constraint: the knots must be in nondecreasing order, with **mu**[**ny** $-$ 4] $>$ **mu**[3].

c – double * *Input*

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

In normal usage, the call to nag_2d_spline_eval (e02dec) follows a call to nag_2d_spline_in terpolant (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 (e02dec).

6: **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.

NE_END_KNOTS_CONS

On entry, the end knots must satisfy $\langle$*value*$\rangle$, $\langle$*value*$\rangle$ = $\langle$*value*$\rangle$, $\langle$*value*$\rangle$ = $\langle$*value*$\rangle$.

NE_INT_ARG_LT

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

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_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*⟩.

7 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.

8 Parallelism and Performance

nag_2d_spline_eval (e02dec) is not threaded in any implementation.

9 Further Comments

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

10 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 a value for m and the coordinates (x_r, y_r), for $r = 1, 2, \ldots, m$, at which the spline is to be evaluated.

10.1 Program Text

```
/* nag_2d_spline_eval (e02dec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

int main(void)
{
  Integer exit_status = 0, i, m;
  NagError fail;
  Nag_2dSpline spline;
  double *ff = 0, *x = 0, *y = 0;

  INIT_FAIL(fail);

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

  printf("nag_2d_spline_eval (e02dec) Example Program Results\n");
#ifdef _WIN32
  scanf_s("%*[^\n]"); /* Skip heading in data file */
#else
  scanf("%*[^\n]"); /* Skip heading in data file */
#endif
  /* Read m, the number of spline evaluation points. */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "", &m);
#else
  scanf("%" NAG_IFMT "", &m);
#endif
  if (m >= 1) {
    if (!(x = NAG_ALLOC(m, double)) ||
        !(y = NAG_ALLOC(m, double)) || !(ff = NAG_ALLOC(m, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  }
  else {
    printf("Invalid m.\n");
    exit_status = 1;
    return exit_status;
  }
  /* Read nx and ny, the number of knots in the x and y directions. */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "", &(spline.nx), &(spline.ny));
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "", &(spline.nx), &(spline.ny));
#endif
  if (spline.nx >= 8 && spline.ny >= 8) {
    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;
    }
  }
  else {
    printf("Invalid spline.nx or spline.ny.\n");
    exit_status = 1;
    return exit_status;
  }

  /* read the knots lamda[0] .. lamda[nx-1] and mu[0] .. mu[ny-1]. */
  for (i = 0; i < spline.nx; i++)
#ifdef _WIN32
    scanf_s("%lf", &(spline.lamda[i]));
#else
    scanf("%lf", &(spline.lamda[i]));
#endif
  for (i = 0; i < spline.ny; i++)
#ifdef _WIN32
    scanf_s("%lf", &(spline.mu[i]));
#else
    scanf("%lf", &(spline.mu[i]));
```

```
#endif
  /* Read c, the bicubic spline coefficients. */
  for (i = 0; i < (spline.nx - 4) * (spline.ny - 4);
#ifdef _WIN32
       scanf_s("%lf", &(spline.c[i])), i++);
#else
       scanf("%lf", &(spline.c[i])), i++);
#endif
  /* Read the x and y co-ordinates of the evaluation points. */
  for (i = 0; i < m; i++)
#ifdef _WIN32
    scanf_s("%lf%lf", &x[i], &y[i]);
#else
    scanf("%lf%lf", &x[i], &y[i]);
#endif
  /* Evaluate the spline at the m points. */
  /* 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;
  }

  /* Print the results. */
  printf("      i          x[i]       y[i]       ff[i]\n");
  for (i = 0; i < m; i++)
    printf("%7" NAG_IFMT "    %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);
END:
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(ff);
  return exit_status;
}
```

10.2 Program Data

```
nag_2d_spline_eval (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
```

10.3 Program Results

```
nag_2d_spline_eval (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
```