NAG Library Function Document

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

3 Description

nag_2d_spline_eval_rect (e02dfc) 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** $- 4$ 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}.

The points in the grid are defined by coordinates x_q, for $q = 1,2,\ldots,m_x$, along the x axis, and coordinates 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 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: **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 axes that define the rectangular grid.

Constraint: **mx** ≥ 1 and **my** ≥ 1.

3: **x**[**mx**] – const double *Input*

4: **y**[**my**] – const double *Input*

On entry: **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 coordinates that define the rectangular grid of points at which values of the spline are required.

Constraint: **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.

5: **ff**[**mx** $\times$ **my**] – double *Output*

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

6: **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 variable 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_rect (e02dfc) 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 (e02dfc).

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_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, **mx** $= \langle value \rangle$.
Constraint: **mx** ≥ 1.

On entry, **my** $= \langle value \rangle$.
Constraint: **my** ≥ 1.

On entry, **spline→nx** must not be less than 8: **spline→nx** $= \langle value \rangle$.

On entry, **spline→ny** must not be less than 8: **spline→ny** $= \langle value \rangle$.

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] $= \langle value \rangle$, **x**[0] $= \langle value \rangle$, **x**[$\langle value \rangle$] $= \langle value \rangle$, **spline→lamda**[$\langle value \rangle$] $= \langle value \rangle$.

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] $= \langle value \rangle$, **y**[0] $= \langle value \rangle$, **y**[$\langle value \rangle$] $= \langle value \rangle$, **spline→mu**[$\langle value \rangle$] $= \langle value \rangle$.

NE_NOT_INCREASING

The sequence **spline→lamda** is not increasing: **spline→lamda**[$\langle value \rangle$] $= \langle value \rangle$, **spline→lamda**[$\langle value \rangle$] $= \langle value \rangle$.
The sequence **spline→mu** is not increasing: **spline→mu**[$\langle value \rangle$] $= \langle value \rangle$, **spline→mu**[$\langle value \rangle$] $= \langle value \rangle$.

NE_NOT_STRICTLY_INCREASING

The sequence **x** is not strictly increasing: **x**[$\langle value \rangle$] $= \langle value \rangle$, **x**[$\langle value \rangle$] $= \langle value \rangle$.
The sequence **y** is not strictly increasing: **y**[$\langle value \rangle$] $= \langle value \rangle$, **y**[$\langle value \rangle$] $= \langle value \rangle$.

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

Not applicable.

9 Further Comments

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

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 values for m_x and the x coordinates x_q, for $q = 1, 2, \ldots, m_x$, and values for m_y and the y coordinates y_r, for $r = 1, 2, \ldots, m_y$, defining the grid of points on which the spline is to be evaluated.

10.1 Program Text

```
/* nag_2d_spline_eval_rect (e02dfc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 * Mark 8 revised, 2004.
 */

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

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

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

  INIT_FAIL(fail);

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

  printf("nag_2d_spline_eval_rect (e02dfc) Example Program Results\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  /* Read mx and my, the number of grid points in the x and y
   * directions respectively.
   */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"", &mx, &my);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"", &mx, &my);
#endif
  if (mx >= 1 && my >= 1)
    {
      if (!(x = NAG_ALLOC(mx, double)) ||
          !(y = NAG_ALLOC(my, double)) ||
          !(ff = NAG_ALLOC(mx*my, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  else
    {
      printf("Invalid mx or my.\n");
      exit_status = 1;
```

```
      return exit_status;
    }
  /* Read spline.nx and spline.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.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("Storage allocation failed.\n");
      exit_status = -1;
      goto END;
    }
  /* 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++)
#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 spline.c, the bicubic spline coefficients. */
  for (i = 0; i < (spline.nx-4)*(spline.ny-4); i++)
#ifdef _WIN32
    scanf_s("%lf", &(spline.c[i]));
#else
    scanf("%lf", &(spline.c[i]));
#endif
  /* Read the x and y co-ordinates defining the evaluation grid. */
  for (i = 0; i < mx;  i++)
#ifdef _WIN32
    scanf_s("%lf", &x[i]);
#else
    scanf("%lf", &x[i]);
#endif
  for (i = 0; i < my;  i++)
#ifdef _WIN32
    scanf_s("%lf", &y[i]);
#else
    scanf("%lf", &y[i]);
#endif
  /* Evaluate the spline at the mx by my points. */
  /* nag_2d_spline_eval_rect (e02dfc).
   * Evaluation of bicubic spline, at a mesh of points
   */
  nag_2d_spline_eval_rect(mx, my, x, y, ff, &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 result array. */
  printf("Spline evaluated on x-y grid (x across, y down):\n      ");
  for (i = 0; i < mx;  i++)
    printf("%2.1f      ", x[i]);
  printf("\n");
```

```
  for (j = 0; j < my; j++)
    {
      printf("%2.1f", y[j]);
      for (i = 0; i < mx; i++)
        printf("%8.3f%s", FF(i, j), (i%7 == 6 || i == mx-1)?"\n":" ");
    }
 END:
  NAG_FREE(spline.c);
  NAG_FREE(spline.lamda);
  NAG_FREE(spline.mu);
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(ff);
  return exit_status;
}
```

10.2 Program Data

```
nag_2d_spline_eval_rect (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
```

10.3 Program Results

```
nag_2d_spline_eval_rect (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
```