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

```c
#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 \( i = 1, 2, \ldots, \text{spline->nx} - 4 \) and \( j = 1, 2, \ldots, \text{spline->ny} - 4 \), 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 \( \lambda_i \) to \( \lambda_{i+4} \) and the latter on the knots \( \mu_j \) to \( \mu_{j+4} \).

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


5 Arguments

1: \( m \) – Integer

\text{Input}

\text{On entry:} \( m \), the number of points at which values of the spline are required.

\text{Constraint:} \( m \geq 1 \).

2: \( x[m] \) – const double

\text{Input}

\text{On entry:} \( x \) 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.

\text{Constraint:} \( x \) and \( y \) must satisfy

\[
spline->lamda[3] \leq x[r - 1] \leq spline->lamda[spline->nx - 4]
\]
n and
The spline representation is not valid outside these intervals, for \( r = 1, 2, \ldots, m \).

\[
spline \rightarrow \text{mu}[3] \leq y[r - 1] \leq \spline \rightarrow \text{mu}[\spline \rightarrow \text{ny} - 4].
\]

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

Pointer to structure of type Nag_2dSpline with the following members:

- \( \text{nx} \) – Integer
  - On entry: \( \text{nx} \) must specify the total number of knots associated with the variables \( x \). It is such that \( \text{nx} - 8 \) is the number of interior knots.
  - Constraint: \( \text{nx} \geq 8 \).

- \( \text{lamda} \) – double *
  - On entry: a pointer to which memory of size \( \text{nx} \) must be allocated. \( \text{lamda} \) must contain the complete sets of knots \( \{ \lambda \} \) associated with the \( x \) variable.
  - Constraint: the knots must be in nondecreasing order, with \( \text{lamda}[\text{nx} - 4] > \text{lamda}[3] \).

- \( \text{ny} \) – Integer
  - On entry: \( \text{ny} \) must specify the total number of knots associated with the variable \( y \).
  - It is such that \( \text{ny} - 8 \) is the number of interior knots.
  - Constraint: \( \text{ny} \geq 8 \).

- \( \text{mu} \) – double *
  - On entry: a pointer to which memory of size \( \text{ny} \) must be allocated. \( \text{mu} \) must contain the complete sets of knots \( \{ \mu \} \) associated with the \( y \) variable.
  - Constraint: the knots must be in nondecreasing order, with \( \text{mu}[\text{ny} - 4] > \text{mu}[3] \).

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

In normal usage, the call to \text{nag_2d_spline_eval (e02dec)} follows a call to \text{nag_2d_spline_interpolant (e01dac)}, \text{nag_2d_spline_fit_grid (e02dcc)} or \text{nag_2d_spline_fit_scat (e02ddc)}, in which case, members of the structure \text{spline} will have been set up correctly for input to \text{nag_2d_spline_eval (e02dec)}.

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

6 Error Indicators and Warnings

\text{NE_ALLOC_FAIL}

Dynamic memory allocation failed.

\text{NE_END_KNOTS_CONS}

On entry, the end knots must satisfy \( \langle \text{value} \rangle, \langle \text{value} \rangle = \langle \text{value} \rangle, \langle \text{value} \rangle = \langle \text{value} \rangle \).
On entry, \( m \geq 1 \).

On entry, \( \text{spline\_nx} \) must not be less than 8: \( \text{spline\_nx} = \langle \text{value} \rangle \).

On entry, \( \text{spline\_ny} \) must not be less than 8: \( \text{spline\_ny} = \langle \text{value} \rangle \).

The sequence \( \text{spline\_lamda} \) is not increasing: \( \text{spline\_lamda} = \langle \text{value} \rangle \).

The sequence \( \text{spline\_mu} \) is not increasing: \( \text{spline\_mu} = \langle \text{value} \rangle \).

On entry, point \( (x, y) = \langle \text{value} \rangle \) lies outside the rectangle bounded by

\( \text{spline\_lamda} = \langle \text{value} \rangle \), \( \text{spline\_mu} = \langle \text{value} \rangle \).

The method used to evaluate the B-splines is numerically stable, in the sense that each computed value of \( s(x, y) \) 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.

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

This program reads in knot sets \( \text{spline\_lamda} = \langle \text{value} \rangle \) and \( \text{spline\_mu} = \langle \text{value} \rangle \), 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

```c
/* nag_2d_spline_eval (e02dec) 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>

int main(void)
{
    Integer exit_status = 0, i, m;
    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 (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(" %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 1.0
1.0000 1.1333 1.3667 1.7000 1.9000 2.0000
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

<table>
<thead>
<tr>
<th>i</th>
<th>x[i]</th>
<th>y[i]</th>
<th>ff[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1</td>
<td>1.100</td>
<td>0.100</td>
<td>1.310</td>
</tr>
<tr>
<td>2</td>
<td>1.500</td>
<td>0.700</td>
<td>2.950</td>
</tr>
<tr>
<td>3</td>
<td>1.600</td>
<td>0.400</td>
<td>2.960</td>
</tr>
<tr>
<td>4</td>
<td>1.900</td>
<td>0.300</td>
<td>3.910</td>
</tr>
<tr>
<td>5</td>
<td>1.900</td>
<td>0.800</td>
<td>4.410</td>
</tr>
<tr>
<td>6</td>
<td>2.000</td>
<td>1.000</td>
<td>5.000</td>
</tr>
</tbody>
</table>