NAG Library Function Document

nag_1d_spline_evaluate (e02bbc)

1 Purpose

nag_1d_spline_evaluate (e02bbc) evaluates a cubic spline from its B-spline representation.

2 Specification

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

void nag_1d_spline_evaluate (double x, double *s, Nag_Spline *spline,
    NagError *fail)
```

3 Description

nag_1d_spline_evaluate (e02bbc) evaluates the cubic spline \( s(x) \) at a prescribed argument \( x \) from its augmented knot set \( \lambda_i \) for \( i = 1, 2, \ldots, n + 7 \), (see nag_1d_spline_fit_knots (e02bac)) and from the coefficients \( c_i \) for \( i = 1, 2, \ldots, q \), in its B-spline representation

\[
s(x) = \sum_{i=1}^{q} c_i N_i(x)
\]

Here \( q = n + 3 \), where \( n \) is the number of intervals of the spline, and \( N_i(x) \) denotes the normalized B-spline of degree 3 defined upon the knots \( \lambda_i, \lambda_{i+1}, \ldots, \lambda_{i+4} \). The prescribed argument \( x \) must satisfy \( \lambda_4 \leq x \leq \lambda_{n+4} \).

It is assumed that \( \lambda_j \geq \lambda_{j-1} \), for \( j = 2, 3, \ldots, n + 7 \), and \( \lambda_{n+4} > \lambda_4 \).

The method employed is that of evaluation by taking convex combinations due to de Boor (1972). For further details of the algorithm and its use see Cox (1972) and Cox (1978).

It is expected that a common use of nag_1d_spline_evaluate (e02bbc) will be the evaluation of the cubic spline approximations produced by nag_1d_spline_fit_knots (e02bac). A generalization of nag_1d_spline_evaluate (e02bbc) which also forms the derivative of \( s(x) \) is nag_1d_spline_deriv (e02bcc). nag_1d_spline_deriv (e02bcc) takes about 50% longer than nag_1d_spline_evaluate (e02bbc).

4 References


Cox M G and Hayes J G (1973) Curve fitting: a guide and suite of algorithms for the non-specialist user NPL Report NAC26 National Physical Laboratory

de Boor C (1972) On calculating with B-splines J. Approx. Theory 6 50–62

5 Arguments

1: \( x \) – double

Input

On entry: the argument \( x \) at which the cubic spline is to be evaluated.

Constraint: \( \text{spline} \rightarrow \text{lamda}[3] \leq x \leq \text{spline} \rightarrow \text{lamda}[\text{spline} \rightarrow n - 4] \).
2: \text{s -- double * } \quad \text{Output}
   \text{On exit: the value of the spline, s(x).}

3: \text{spline -- Nag_Spline *}
   \text{Pointer to structure of type Nag_Spline with the following members:}
   \begin{itemize}
   \item \text{n -- Integer } \quad \text{Input}
     \text{On entry: } \bar{n} + 7, \text{ where } \bar{n} \text{ is the number of intervals (one greater than the number of interior knots, i.e., the knots strictly within the range } \lambda_4 \text{ to } \lambda_{\bar{n}+4} \text{ over which the spline is defined.}
     \text{Constraint: } \text{spline} \rightarrow n \geq 8.
   \item \text{lamda -- double * } \quad \text{Input}
     \text{On entry: a pointer to which memory of size spline} \rightarrow n \text{ must be allocated. spline} \rightarrow \text{lamda}[j-1] \text{ must be set to the value of the } j \text{th member of the complete set of knots, } \lambda_j \text{ for } j = 1, 2, \ldots, \bar{n} + 7.
     \text{Constraint: the } \lambda_j \text{ must be in nondecreasing order with spline} \rightarrow \text{lamda} [\text{spline} \rightarrow n - 4] > \text{spline} \rightarrow \text{lamda}[3].
   \item \text{c -- double * } \quad \text{Input}
     \text{On entry: a pointer to which memory of size spline} \rightarrow n - 4 \text{ must be allocated. spline} \rightarrow c \text{ holds the coefficient } c_i \text{ of the B-spline } N_i(x), \text{ for } i = 1, 2, \ldots, \bar{n} + 3.
   \end{itemize}

   Under normal usage, the call to nag_1d_spline_evaluate (e02bbc) will follow a call to nag_1d_spline_fit_knots (e02bac), nag_1d_spline_interpolant (e01bac) or nag_1d_spline_fit (e02bec). In that case, the structure spline will have been set up correctly for input to nag_1d_spline_evaluate (e02bbc).

4: \text{fail -- NagError * } \quad \text{Input/Output}
   \text{The NAG error argument (see Section 3.6 in the Essential Introduction).}

6 \quad \text{Error Indicators and Warnings}

\text{NE_ABSCI_OUTSIDE_KNOT_INVL}
   \text{On entry, x must satisfy spline} \rightarrow \text{lamda}[3] \leq x \leq \text{spline} \rightarrow \text{lamda}[\text{spline} \rightarrow n - 4]:
   \text{spline} \rightarrow \text{lamda}[3] = \langle \text{value} \rangle, \text{ x = } \langle \text{value} \rangle, \text{spline} \rightarrow \text{lamda}[\langle \text{value} \rangle] = \langle \text{value} \rangle.
   \text{In this case } \text{s is set arbitrarily to zero.}

\text{NE_INT_ARG_LT}
   \text{On entry, spline} \rightarrow n \text{ must not be less than 8: } \text{spline} \rightarrow n = \langle \text{value} \rangle.

7 \quad \text{Accuracy}

The computed value of \( s(x) \) has negligible error in most practical situations. Specifically, this value has an absolute error bounded in modulus by \( 18 \times c_{\text{max}} \times \text{machine precision} \), where \( c_{\text{max}} \) is the largest in modulus of \( c_j, c_{j+1}, c_{j+2} \text{ and } c_{j+3} \), and \( j \) is an integer such that \( \lambda_{j+3} \leq x \leq \lambda_{j+4} \). If \( c_j, c_{j+1}, c_{j+2} \text{ and } c_{j+3} \) are all of the same sign, then the computed value of \( s(x) \) has a relative error not exceeding \( 20 \times \text{machine precision} \) in modulus. For further details see Cox (1978).

8 \quad \text{Parallelism and Performance}

Not applicable.
9 Further Comments

The time taken by nag_1d_spline_evaluate (e02bbc) is approximately \( C \times (1 + 0.1 \times \log (\tilde{n} + 7)) \) seconds, where \( C \) is a machine-dependent constant.

Note: the function does not test all the conditions on the knots given in the description of \( \text{spline} \rightarrow \lambda \) in Section 5, since to do this would result in a computation time approximately linear in \( \log (\tilde{n} + 7) \). All the conditions are tested in nag_1d_spline_fit_knots (e02bac), however, and the knots returned by nag_1d_spline_interpolant (e01bac) or nag_1d_spline_fit (e02bec) will satisfy the conditions.

10 Example

Evaluate at 9 equally-spaced points in the interval \( 1.0 \leq x \leq 9.0 \) the cubic spline with (augmented) knots 1.0, 1.0, 1.0, 1.0, 3.0, 6.0, 8.0, 9.0, 9.0, 9.0, 9.0 and normalized cubic B-spline coefficients 1.0, 2.0, 4.0, 7.0, 6.0, 4.0, 3.0.

The example program is written in a general form that will enable a cubic spline with \( \tilde{n} \) intervals, in its normalized cubic B-spline form, to be evaluated at \( m \) equally-spaced points in the interval \( \text{spline} \rightarrow \lambda \) \( 3 \leq x \leq \text{spline} \rightarrow \lambda \), \( \tilde{n} + 3 \). The program is self-starting in that any number of datasets may be supplied.

10.1 Program Text

/* nag_1d_spline_evaluate (e02bbc) Example Program. *
 * Copyright 2014 Numerical Algorithms Group.
 * Mark 3 revised, 1994.
 */
#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nage02.h>

int main(void)
{
    Integer exit_status = 0, j, m, ncap, ncap7, r;
    Nag_Spline spline;
    double a, b, s, x;
    NagError fail;

    INIT_FAIL(fail);

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

    printf("nag_1d_spline_evaluate (e02bbc) Example Program Results\n");
#if defined _WIN32
    scanf_s("%*[\n]"); /* Skip heading in data file */
#else
    scanf("%*[\n]"); /* Skip heading in data file */
#endif
#if defined _WIN32
    while (scanf_s("%NAG_IFMT", &m) != EOF)
#else
    while (scanf("%NAG_IFMT", &m) != EOF)
#endif
    {
        if (m <= 0)
        {
            printf("Invalid m.\n");
            exit_status = 1;
        }
return exit_status;
}
#endif
scanf_s("%"NAG_IFMT", &ncap);
#else
scanf("%"NAG_IFMT", &ncap);
#endif
ncap7 = ncap+7;
if (ncap > 0)
{
    spline.n = ncap7;
    if ((!spline.c = NAG_ALLOC(ncap7, double)) || (!spline.lamda = NAG_ALLOC(ncap7, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    for (j = 0; j < ncap7; j++)
    {
        #ifdef _WIN32
        scanf_s("%lf", &(spline.lamda[j]));
        #else
        scanf("%lf", &(spline.lamda[j]));
        #endif
    }
    for (j = 0; j < ncap+3; j++)
    {
        #ifdef _WIN32
        scanf_s("%lf", &(spline.c[j]));
        #else
        scanf("%lf", &(spline.c[j]));
        #endif
    }
    a = spline.lamda[3];
    b = spline.lamda[ncap+3];
    printf("Augmented set of knots stored in spline.lamda:\n");
    for (j = 0; j < ncap7; j++)
        printf("%10.4f%s", spline.lamda[j], (j%6 == 5 || j == ncap7-1) ? "\n": " ");
    printf("\nB-spline coefficients stored in spline.c\n");
    for (j = 0; j < ncap+3; j++)
        printf("%10.4f", spline.c[j]);
    printf("\n x Value of cubic spline\n");
    for (r = 1; r <= m; ++r)
    {
        x = ((double)(m-r) * a + (double)(r-1) * b) / (double)(m-1);
        /* nag_1d_spline_evaluate (e02bbc). */
        nag_1d_spline_evaluate(x, &s, &spline, &fail);
        if (fail.code != NE_NOERROR)
        {
            printf("Error from nag_1d_spline_evaluate (e02bbc).\n%s\n", fail.message);
            exit_status = 1;
            goto END;
        }
        printf("%10.4f%15.4f\n", x, s);
    }
    NAG_FREE(spline.c);
    NAG_FREE(spline.lamda);
}
END:
return exit_status;
}
10.2 Program Data

nag_1d_spline_evaluate (e02bbc) Example Program Data

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>3.00</td>
<td>6.00</td>
</tr>
<tr>
<td>8.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
<td>9.00</td>
</tr>
<tr>
<td>1.00</td>
<td>2.00</td>
<td>4.00</td>
<td>7.00</td>
<td>6.00</td>
</tr>
<tr>
<td>4.00</td>
<td>3.00</td>
<td>2.00</td>
<td>4.00</td>
<td>7.00</td>
</tr>
<tr>
<td>3.00</td>
<td>1.00</td>
<td>2.00</td>
<td>4.00</td>
<td>7.00</td>
</tr>
</tbody>
</table>

10.3 Program Results

nag_1d_spline_evaluate (e02bbc) Example Program Results

Augmented set of knots stored in spline.lamda:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>3.0000</td>
<td>6.0000</td>
</tr>
<tr>
<td>9.0000</td>
<td>9.0000</td>
<td>9.0000</td>
<td>9.0000</td>
<td>9.0000</td>
<td>6.0000</td>
</tr>
</tbody>
</table>

B-spline coefficients stored in spline.c

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000</td>
<td>2.0000</td>
<td>4.0000</td>
<td>7.0000</td>
<td>6.0000</td>
<td>4.0000</td>
</tr>
<tr>
<td>3.0000</td>
<td>4.0000</td>
<td>7.0000</td>
<td>4.0000</td>
<td>7.0000</td>
<td>4.0000</td>
</tr>
</tbody>
</table>

x Value of cubic spline

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>2.0000</td>
<td>2.3779</td>
</tr>
<tr>
<td>3.0000</td>
<td>3.6229</td>
</tr>
<tr>
<td>4.0000</td>
<td>4.8327</td>
</tr>
<tr>
<td>5.0000</td>
<td>5.8273</td>
</tr>
<tr>
<td>6.0000</td>
<td>6.3571</td>
</tr>
<tr>
<td>7.0000</td>
<td>6.1905</td>
</tr>
<tr>
<td>8.0000</td>
<td>5.1667</td>
</tr>
<tr>
<td>9.0000</td>
<td>3.0000</td>
</tr>
</tbody>
</table>