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

```
#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 λ_i, for $i = 1, 2, \ldots, \bar{n} + 7$, (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 $q = \bar{n} + 3$, where $\bar{n}$ 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 $\lambda_4 \leq x \leq \lambda_{\bar{n}+4}$.

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

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 (1972) The numerical evaluation of B-splines *J. Inst. Math. Appl.* **10** 134–149

Cox M G (1978) The numerical evaluation of a spline from its B-spline representation *J. Inst. Math. Appl.* **21** 135–143

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: **spline→lamda**$[3] \leq$ **x** $\leq$ **spline→lamda**[**spline→n** $- 4]$.

2: **s** – double * *Output*

On exit: the value of the spline, $s(x)$.

3: **spline** – Nag_Spline *

Pointer to structure of type Nag_Spline with the following members:

n – Integer *Input*

On entry: $\bar{n}+7$, where $\bar{n}$ is the number of intervals (one greater than the number of interior knots, i.e., the knots strictly within the range λ_4 to $\lambda_{\bar{n}+4}$) over which the spline is defined.

Constraint: **spline→n** ≥ 8.

lamda – double * *Input*

On entry: a pointer to which memory of size **spline→n** must be allocated. **spline→lamda**$[j-1]$ must be set to the value of the jth member of the complete set of knots, λ_j for $j=1,2,\ldots,\bar{n}+7$.

Constraint: the λ_j must be in nondecreasing order with **spline→lamda**[**spline→n** $-$ 4] $>$ **spline→lamda**[3].

c – double * *Input*

On entry: a pointer to which memory of size **spline→n** $-$ 4 must be allocated. **spline→c** holds the coefficient c_i of the B-spline $N_i(x)$, for $i=1,2,\ldots,\bar{n}+3$.

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: **fail** – NagError * *Input/Output*

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

6 Error Indicators and Warnings

NE_ABSCI_OUTSIDE_KNOT_INTVL

On entry, **x** must satisfy **spline→lamda**[3] $\leq$ **x** $\leq$ **spline→lamda**[**spline→n** $-$ 4]: **spline→lamda**[3] $= \langle value \rangle$, **x** $= \langle value \rangle$, **spline→lamda**[$\langle value \rangle$] $= \langle value \rangle$.
In this case **s** is set arbitrarily to zero.

NE_INT_ARG_LT

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

7 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 $18 \times c_{\max} \times$ ***machine precision***, where $c_{\max}$ is the largest in modulus of c_j, c_{j+1}, c_{j+2} 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} and c_{j+3} are all of the same sign, then the computed value of $s(x)$ has a relative error not exceeding $20\times$ ***machine precision*** in modulus. For further details see Cox (1978).

8 Parallelism and Performance

Not applicable.

9 Further Comments

The time taken by nag_1d_spline_evaluate (e02bbc) is approximately C $\times(1 + 0.1 \times \log(\bar{n} + 7))$ 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 **spline→lamda** in Section 5, since to do this would result in a computation time approximately linear in $\bar{n} + 7$ instead of $\log(\bar{n} + 7)$. 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 $\bar{n}$ intervals, in its normalized cubic B-spline form, to be evaluated at m equally-spaced points in the interval **spline→lamda**$[3] \leq x \leq$ **spline→lamda**$[\bar{n} + 3]$. 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 2, 1991.
 *
 * Mark 3 revised, 1994.
 * Mark 8 revised, 2004.
 */

#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");
#ifdef _WIN32
  scanf_s("%*[^\n]");  /* Skip heading in data file */
#else
  scanf("%*[^\n]");  /* Skip heading in data file */
#endif
#ifdef _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;
        }
#ifdef _WIN32
      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;
            }
        }
      else
        {
          printf("Invalid ncap.\n");
          exit_status = 1;
          return exit_status;
        }
      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\n");
      for (j = 0; j < ncap+3; j++)
        printf("%10.4f%s", spline.c[j],
                (j%6 == 5 || j == ncap+2)?"\n":" ");
      printf("\n      x         Value of cubic spline\n\n");
      for (r = 1; r <= m; ++r)
        {
          x = ((double)(m-r) * a + (double)(r-1) * b) / (double)(m-1);
          /* nag_1d_spline_evaluate (e02bbc).
           * Evaluation of fitted cubic spline, function only
           */
          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
  9
    4
      1.00
      1.00
      1.00
      1.00
      3.00
      6.00
      8.00
      9.00
      9.00
      9.00
      9.00
      1.00
      2.00
      4.00
      7.00
      6.00
      4.00
      3.00
```

10.3 Program Results

```
nag_1d_spline_evaluate (e02bbc) Example Program Results
Augmented set of knots stored in spline.lamda:
    1.0000     1.0000     1.0000     1.0000     3.0000     6.0000
    8.0000     9.0000     9.0000     9.0000     9.0000

B-spline coefficients stored in spline.c

    1.0000     2.0000     4.0000     7.0000     6.0000     4.0000
    3.0000

      x         Value of cubic spline

    1.0000        1.0000
    2.0000        2.3779
    3.0000        3.6229
    4.0000        4.8327
    5.0000        5.8273
    6.0000        6.3571
    7.0000        6.1905
    8.0000        5.1667
    9.0000        3.0000
```