

# NAG Library Function Document

## nag\_1d\_spline\_deriv (e02bcc)

### 1 Purpose

nag\_1d\_spline\_deriv (e02bcc) evaluates a cubic spline and its first three derivatives from its B-spline representation.

### 2 Specification

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

void nag_1d_spline_deriv (Nag_DerivType derivs, double x, double s[],
    Nag_Spline *spline, NagError *fail)
```

### 3 Description

nag\_1d\_spline\_deriv (e02bcc) evaluates the cubic spline  $s(x)$  and its first three derivatives at a prescribed argument  $x$ . It is assumed that  $s(x)$  is represented in terms of its B-spline coefficients  $c_i$ , for  $i = 1, 2, \dots, \bar{n} + 3$  and (augmented) ordered knot set  $\lambda_i$ , for  $i = 1, 2, \dots, \bar{n} + 7$ , (see nag\_1d\_spline\_fit\_knots (e02bac)), i.e.,

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

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

At a simple knot  $\lambda_i$  (i.e., one satisfying  $\lambda_{i-1} < \lambda_i < \lambda_{i+1}$ ), the third derivative of the spline is in general discontinuous. At a multiple knot (i.e., two or more knots with the same value), lower derivatives, and even the spline itself, may be discontinuous. Specifically, at a point  $x = u$  where (exactly)  $r$  knots coincide (such a point is termed a knot of multiplicity  $r$ ), the values of the derivatives of order  $4 - j$ , for  $j = 1, 2, \dots, r$ , are in general discontinuous. (Here  $1 \leq r \leq 4$ ;  $r > 4$  is not meaningful.) You must specify whether the value at such a point is required to be the left- or right-hand derivative.

The method employed is based upon:

- (i) carrying out a binary search for the knot interval containing the argument  $x$  (see Cox (1978)),
- (ii) evaluating the nonzero B-splines of orders 1, 2, 3 and 4 by recurrence (see Cox (1972) and Cox (1978)),
- (iii) computing all derivatives of the B-splines of order 4 by applying a second recurrence to these computed B-spline values (see de Boor (1972)),
- (iv) multiplying the 4th-order B-spline values and their derivative by the appropriate B-spline coefficients, and summing, to yield the values of  $s(x)$  and its derivatives.

nag\_1d\_spline\_deriv (e02bcc) can be used to compute the values and derivatives of cubic spline fits and interpolants produced by nag\_1d\_spline\_fit\_knots (e02bac), nag\_1d\_spline\_fit (e02bec) or nag\_1d\_spline\_interpolant (e01bac).

If only values and not derivatives are required, nag\_1d\_spline\_evaluate (e02bbc) may be used instead of nag\_1d\_spline\_deriv (e02bcc), which 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
- de Boor C (1972) On calculating with B-splines *J. Approx. Theory* **6** 50–62

## 5 Arguments

- 1: **derivs** – Nag\_DerivType *Input*  
*On entry:* **derivs**, of type Nag\_DerivType, specifies whether left- or right-hand values of the spline and its derivatives are to be computed (see Section 3). Left- or right-hand values are formed according to whether **derivs** is equal to Nag\_LeftDerivs or Nag\_RightDerivs respectively. If  $x$  does not coincide with a knot, the value of **derivs** is immaterial. If  $x = \mathbf{spline} \rightarrow \mathbf{lamda}[3]$ , right-hand values are computed, and if  $x = \mathbf{spline} \rightarrow \mathbf{lamda}[\mathbf{spline} \rightarrow \mathbf{n} - 4]$ , left-hand values are formed, regardless of the value of **derivs**.  
*Constraint:* **derivs** = Nag\_LeftDerivs or Nag\_RightDerivs.
- 2: **x** – double *Input*  
*On entry:* the argument  $x$  at which the cubic spline and its derivatives are to be evaluated.  
*Constraint:*  $\mathbf{spline} \rightarrow \mathbf{lamda}[3] \leq x \leq \mathbf{spline} \rightarrow \mathbf{lamda}[\mathbf{spline} \rightarrow \mathbf{n} - 4]$ .
- 3: **s[4]** – double *Output*  
*On exit:* **s[j]** contains the value of the  $j$ th derivative of the spline at the argument  $x$ , for  $j = 0, 1, 2, 3$ . Note that **s[0]** contains the value of the spline.
- 4: **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 of the spline (which is one greater than the number of interior knots, i.e., the knots strictly within the range  $\lambda_4$  to  $\lambda_{\bar{n}+4}$  over which the spline is defined).  
*Constraint:*  $\mathbf{spline} \rightarrow \mathbf{n} \geq 8$ .

**lamda** – double *Input*  
*On entry:* a pointer to which memory of size  $\mathbf{spline} \rightarrow \mathbf{n}$  must be allocated.  $\mathbf{spline} \rightarrow \mathbf{lamda}[j - 1]$  must be set to the value of the  $j$ th member of the complete set of knots,  $\lambda_j$ , for  $j = 1, 2, \dots, \bar{n} + 7$ .  
*Constraint:* the  $\lambda_j$  must be in nondecreasing order with  $\mathbf{spline} \rightarrow \mathbf{lamda}[\mathbf{spline} \rightarrow \mathbf{n} - 4] > \mathbf{spline} \rightarrow \mathbf{lamda}[3]$ .

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

Under normal usage, the call to nag\_1d\_spline\_deriv (e02bcc) 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\_deriv (e02bcc).

5: **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\_ABSCL\_OUTSIDE\_KNOT\_INTVL

On entry,  $\mathbf{spline} \rightarrow \mathbf{lamda}[3] \leq \mathbf{x} \leq \mathbf{spline} \rightarrow \mathbf{lamda}[\mathbf{spline} \rightarrow \mathbf{n} - 4]$ :  
 $\mathbf{spline} \rightarrow \mathbf{lamda}[3] = \langle \text{value} \rangle$ ,  $\mathbf{x} = \langle \text{value} \rangle$ ,  $\mathbf{spline} \rightarrow \mathbf{lamda}[\langle \text{value} \rangle] = \langle \text{value} \rangle$ .

### NE\_BAD\_PARAM

On entry, argument **derivs** had an illegal value.

### NE\_INT\_ARG\_LT

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

### NE\_SPLINE\_RANGE\_INVALID

On entry, the cubic spline range is invalid:

$\mathbf{spline} \rightarrow \mathbf{lamda}[3] = \langle \text{value} \rangle$  while  $\mathbf{spline} \rightarrow \mathbf{lamda}[\mathbf{spline} \rightarrow \mathbf{n} - 4] = \langle \text{value} \rangle$ .  
 These must satisfy  $\mathbf{spline} \rightarrow \mathbf{lamda}[3] < \mathbf{spline} \rightarrow \mathbf{lamda}[\mathbf{spline} \rightarrow \mathbf{n} - 4]$ .

## 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  $18 \times c_{\max} \times \text{machine precision}$ , 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 relative error bounded by  $20 \times \text{machine precision}$ . For full details see Cox (1978).

No complete error analysis is available for the computation of the derivatives of  $s(x)$ . However, for most practical purposes the absolute errors in the computed derivatives should be small.

## 8 Parallelism and Performance

nag\_1d\_spline\_deriv (e02bcc) is not threaded in any implementation.

## 9 Further Comments

The time taken by this function is approximately linear in  $\log(\bar{n} + 7)$ .

Note: the function does not test all the conditions on the knots given in the description of  $\mathbf{spline} \rightarrow \mathbf{lamda}$  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

Compute, at the 7 arguments  $x = 0, 1, 2, 3, 4, 5, 6$ , the left- and right-hand values and first 3 derivatives of the cubic spline defined over the interval  $0 \leq x \leq 6$  having the 6 interior knots  $x = 1, 3, 3, 3, 4, 4$ , the 8 additional knots  $0, 0, 0, 0, 6, 6, 6, 6$ , and the 10 B-spline coefficients 10, 12, 13, 15, 22, 26, 24, 18, 14, 12.

The input data items (using the notation of Section 5) comprise the following values in the order indicated:

$\bar{n}$	$m$
<b>spline</b> → <b>lamda</b> [ $j$ ]	for $j = 0, 1, \dots, \bar{n} + 6$
<b>spline</b> → <b>c</b> [ $j$ ],	for $j = 0, 1, \dots, \bar{n} + 2$
<b>x</b>	$m$ values of <b>x</b>

The example program is written in a general form that will enable the values and derivatives of a cubic spline having an arbitrary number of knots to be evaluated at a set of arbitrary points. Any number of datasets may be supplied.

## 10.1 Program Text

```

/* nag_ld_spline_deriv (e02bcc) 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, j, l, m, ncap, ncap7;
    NagError fail;
    Nag_DerivType derivs;
    Nag_Spline spline;
    double s[4], x;

    INIT_FAIL(fail);

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

    printf("nag_ld_spline_deriv (e02bcc) 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 "%" NAG_IFMT "", &ncap, &m) != EOF)
#else
    while (scanf("%" NAG_IFMT "%" NAG_IFMT "", &ncap, &m) != EOF)
#endif
    {
        if (m <= 0) {
            printf("Invalid m.\n");
            exit_status = 1;
            return exit_status;
        }
        if (ncap > 0) {
            ncap7 = ncap + 7;
            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
        printf("          x          Spline      1st deriv  "
               "2nd deriv  3rd deriv");
        for (i = 1; i <= m; i++) {
#ifdef _WIN32
            scanf_s("%lf", &x);
#else
            scanf("%lf", &x);
#endif
            derivs = Nag_LeftDerivs;
            for (j = 1; j <= 2; j++) {
                /* nag_ld_spline_deriv (e02bcc).
                 * Evaluation of fitted cubic spline, function and
                 * derivatives
                 */
                nag_ld_spline_deriv(derivs, x, s, &spline, &fail);
                if (fail.code != NE_NOERROR) {
                    printf("Error from nag_ld_spline_deriv (e02bcc).\n%s\n",
                           fail.message);
                    exit_status = 1;
                    goto END;
                }

                if (derivs == Nag_LeftDerivs) {
                    printf("\n\n%11.4f  Left", x);
                    for (l = 0; l < 4; l++)
                        printf("%11.4f", s[l]);
                }
                else {
                    printf("\n\n%11.4f  Right", x);
                    for (l = 0; l < 4; l++)
                        printf("%11.4f", s[l]);
                }
                derivs = Nag_RightDerivs;
            }
        }
        printf("\n");
    END:
        NAG_FREE(spline.c);
        NAG_FREE(spline.lamda);
    }
    return exit_status;
}

```

## 10.2 Program Data

nag\_ld\_spline\_deriv (e02bcc) Example Program Data

7	7						
0.0	0.0	0.0	0.0	1.0	3.0	3.0	3.0
4.0	4.0	6.0	6.0	6.0	6.0		
10.0	12.0	13.0	15.0	22.0	26.0	24.0	18.0
14.0	12.0						
0.0							
1.0							

2.0  
3.0  
4.0  
5.0  
6.0

### 10.3 Program Results

nag_1d_spline_deriv (e02bcc) Example Program Results					
x		Spline	1st deriv	2nd deriv	3rd deriv
0.0000	Left	10.0000	6.0000	-10.0000	10.6667
0.0000	Right	10.0000	6.0000	-10.0000	10.6667
1.0000	Left	12.7778	1.3333	0.6667	10.6667
1.0000	Right	12.7778	1.3333	0.6667	3.9167
2.0000	Left	15.0972	3.9583	4.5833	3.9167
2.0000	Right	15.0972	3.9583	4.5833	3.9167
3.0000	Left	22.0000	10.5000	8.5000	3.9167
3.0000	Right	22.0000	12.0000	-36.0000	36.0000
4.0000	Left	22.0000	-6.0000	0.0000	36.0000
4.0000	Right	22.0000	-6.0000	0.0000	1.5000
5.0000	Left	16.2500	-5.2500	1.5000	1.5000
5.0000	Right	16.2500	-5.2500	1.5000	1.5000
6.0000	Left	12.0000	-3.0000	3.0000	1.5000
6.0000	Right	12.0000	-3.0000	3.0000	1.5000

---