

# NAG Library Function Document

## nag\_monotonic\_interpolant (e01bec)

### 1 Purpose

nag\_monotonic\_interpolant (e01bec) computes a monotonicity-preserving piecewise cubic Hermite interpolant to a set of data points.

### 2 Specification

```
#include <nag.h>
#include <nage01.h>

void nag_monotonic_interpolant (Integer n, const double x[],
                                const double f[], double d[], NagError *fail)
```

### 3 Description

nag\_monotonic\_interpolant (e01bec) estimates first derivatives at the set of data points  $(x_r, f_r)$ , for  $r = 0, 1, \dots, n-1$ , which determine a piecewise cubic Hermite interpolant to the data, that preserves monotonicity over ranges where the data points are monotonic. If the data points are only piecewise monotonic, the interpolant will have an extremum at each point where monotonicity switches direction. The estimates of the derivatives are computed by a formula due to Brodlie, which is described in Fritsch and Butland (1984), with suitable changes at the boundary points.

The algorithm is derived from routine PCHIM in Fritsch (1982).

Values of the computed interpolant can subsequently be computed by calling nag\_monotonic\_evaluate (e01bec).

### 4 References

Fritsch F N (1982) PCHIP final specifications *Report UCID-30194* Lawrence Livermore National Laboratory

Fritsch F N and Butland J (1984) A method for constructing local monotone piecewise cubic interpolants *SIAM J. Sci. Statist. Comput.* **5** 300–304

### 5 Arguments

- 1: **n** – Integer *Input*  
*On entry:*  $n$ , the number of data points.  
*Constraint:*  $n \geq 2$ .
- 2: **x[n]** – const double *Input*  
*On entry:* **x[r]** must be set to  $x_r$ , the  $r$ th value of the independent variable (abscissa), for  $r = 0, 1, \dots, n-1$ .  
*Constraint:*  $x[r] < x[r+1]$ .
- 3: **f[n]** – const double *Input*  
*On entry:* **f[r]** must be set to  $f_r$ , the  $r$ th value of the dependent variable (ordinate), for  $r = 0, 1, \dots, n-1$ .

- 4: **d[n]** – double *Output*  
*On exit:* estimates of derivatives at the data points. **d[r]** contains the derivative at **x[r]**.
- 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\_INT\_ARG\_LT

On entry, **n** =  $\langle value \rangle$ .  
 Constraint: **n**  $\geq 2$ .

### NE\_NOT\_MONOTONIC

On entry, **x[r - 1]**  $\geq$  **x[r]** for  $r = \langle value \rangle$ : **x[r - 1]**, **x[r]** =  $\langle values \rangle$ .  
 The values of **x[r]**, for  $r = 0, 1, \dots, n - 1$ , are not in strictly increasing order.

## 7 Accuracy

The computational errors in the array **d** should be negligible in most practical situations.

## 8 Parallelism and Performance

nag\_monotonic\_interpolant (e01bec) is not threaded in any implementation.

## 9 Further Comments

The time taken by nag\_monotonic\_interpolant (e01bec) is approximately proportional to  $n$ .

The values of the computed interpolant at the points  $[i]$ , for  $i = 0, 1, \dots, m - 1$ , may be obtained in the real array **pf**, of length at least  $m$ , by the call:

```
e01bfc (n, x, f, d, m, px, pf, &fail)
```

where **n**, **x** and **f** are the input arguments to nag\_monotonic\_interpolant (e01bec) and **d** is the output argument from nag\_monotonic\_interpolant (e01bec).

## 10 Example

This example program reads in a set of data points, calls nag\_monotonic\_interpolant (e01bec) to compute a piecewise monotonic interpolant, and then calls nag\_monotonic\_evaluate (e01bfc) to evaluate the interpolant at equally spaced points.

### 10.1 Program Text

```
/* nag_monotonic_interpolant (e01bec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group
 *
 * Mark 26, 2016.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nage01.h>
```

```

int main(void)
{
    Integer exit_status = 0, i, m, n, r;
    NagError fail;
    double *d = 0, *f = 0, *pf = 0, *px = 0, step, *x = 0;

    INIT_FAIL(fail);

    printf("nag_monotonic_interpolant (e01bec) Example Program Results\n");
#ifdef _WIN32
    scanf_s("%*[^\\n]"); /* Skip to end of line */
#else
    scanf("%*[^\\n]"); /* Skip to end of line */
#endif
#ifdef _WIN32
    scanf_s("%" NAG_IFMT " ", &n);
#else
    scanf("%" NAG_IFMT " ", &n);
#endif
    if (n >= 2) {
        if (!(d = NAG_ALLOC(n, double)) ||
            !(f = NAG_ALLOC(n, double)) || !(x = NAG_ALLOC(n, double)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
    else {
        printf("Invalid n.\n");
        exit_status = 1;
        return exit_status;
    }
    for (r = 0; r < n; r++)
#ifdef _WIN32
        scanf_s("%lf%lf", &x[r], &f[r]);
#else
        scanf("%lf%lf", &x[r], &f[r]);
#endif
    /* Abort on error in nag_monotonic_interpolant (e01bec) */
    /* nag_monotonic_interpolant (e01bec).
     * Interpolating function, monotonicity-preserving,
     * piecewise cubic Hermite, one variable
     */
    nag_monotonic_interpolant(n, x, f, d, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_monotonic_interpolant (e01bec).\n%s\n",
            fail.message);
        exit_status = 1;
        goto END;
    }
#ifdef _WIN32
    scanf_s("%" NAG_IFMT " ", &m);
#else
    scanf("%" NAG_IFMT " ", &m);
#endif
    if (m >= 1) {
        if (!(pf = NAG_ALLOC(m, double)) || !(px = NAG_ALLOC(m, double)))
        {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }
    }
    else {
        printf("Invalid m.\n");
        exit_status = 1;
        return exit_status;
    }
    /* Compute M equally spaced points from x[0] to x[n-1]. */
    step = (x[n - 1] - x[0]) / (double) (m - 1);

```

```

    for (i = 0; i < m; i++)
        px[i] = MIN(x[0] + i * step, x[n - 1]);
    /* nag_monotonic_evaluate (e01bfc).
    * Evaluation of interpolant computed by
    * nag_monotonic_interpolant (e01bec), function only
    */
    nag_monotonic_evaluate(n, x, f, d, m, px, pf, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_monotonic_evaluate (e01bfc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
    printf("
                Interpolated\n");
    printf("
    Abscissa      Value\n");
    for (i = 0; i < m; i++)
        printf("%13.4f%13.4f\n", px[i], pf[i]);
END:
    NAG_FREE(d);
    NAG_FREE(f);
    NAG_FREE(pf);
    NAG_FREE(px);
    NAG_FREE(x);
    return exit_status;
}

```

## 10.2 Program Data

nag\_monotonic\_interpolant (e01bec) Example Program Data

```

9
7.99  0.00000E+0
8.09  0.27643E-4
8.19  0.43750E-1
8.70  0.16918E+0
9.20  0.46943E+0
10.00 0.94374E+0
12.00 0.99864E+0
15.00 0.99992E+0
20.00 0.99999E+0
11

```

## 10.3 Program Results

nag\_monotonic\_interpolant (e01bec) Example Program Results

Abcissa	Interpolated Value
7.9900	0.0000
9.1910	0.4640
10.3920	0.9645
11.5930	0.9965
12.7940	0.9992
13.9950	0.9998
15.1960	0.9999
16.3970	1.0000
17.5980	1.0000
18.7990	1.0000
20.0000	1.0000

---