NAG Library Function Document

nag_monotonic_evaluate (e01bfc)

1 Purpose

nag_monotonic_evaluate (e01bfc) evaluates a piecewise cubic Hermite interpolant at a set of points.

2 Specification

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

void nag_monotonic_evaluate (Integer n, const double x[], const double f[],
const double d[], Integer m, const double px[], double pf[],
NagError *fail)

3 Description

A piecewise cubic Hermite interpolant, as computed by nag_monotonic_interpolant (e01bec), is evaluated at the points \( px[i] \), for \( i = 0, 1, \ldots, m - 1 \). If any point lies outside the interval from \( x[0] \) to \( x[n - 1] \), a value is extrapolated from the nearest extreme cubic, and a warning is returned.

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

4 References


5 Arguments

1: \( n \) – Integer

\( n \) must be unchanged from the previous call of nag_monotonic_interpolant (e01bec).

2: \( x[n] \) – const double

\( x[n] \) must be unchanged from the previous call of nag_monotonic_interpolant (e01bec).

3: \( f[n] \) – const double

\( f[n] \) must be unchanged from the previous call of nag_monotonic_interpolant (e01bec).

4: \( d[n] \) – const double

\( d[n] \) must be unchanged from the previous call of nag_monotonic_interpolant (e01bec).

5: \( m \) – Integer

\( m \), the number of points at which the interpolant is to be evaluated.

\( m \geq 1 \).

6: \( px[m] \) – const double

\( px[m] \) is the \( m \) values of \( x \) at which the interpolant is to be evaluated.

7: \( pf[m] \) – double

\( pf[m] \) contains the value of the interpolant evaluated at the point \( px[i] \), for \( i = 0, 1, \ldots, m - 1 \).
6 Error Indicators and Warnings

**NE_INT_ARG_LT**

On entry, \( m = \langle \text{value} \rangle \).
Constraint: \( m \geq 1 \).

On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 2 \).

**NE_NOT_MONOTONIC**

On entry, \( x[r-1] \geq x[r] \) for \( r = \langle \text{value} \rangle \): \( x[r-1], x[r] = \langle \text{values} \rangle \).
The values of \( x[r] \), for \( r = 0, 1, \ldots, n-1 \), are not in strictly increasing order.

**NW_EXTRAPOLATE**

Warning – some points in array PX lie outside the range \( x[0], \ldots, x[n-1] \). Values at these points are unreliable as they have been computed by extrapolation.

7 Accuracy

The computational errors in the array \( pf \) should be negligible in most practical situations.

8 Parallelism and Performance

Not applicable.

9 Further Comments

The time taken by \( \text{nag\_monotonic\_evaluate} \) is approximately proportional to the number of evaluation points, \( m \). The evaluation will be most efficient if the elements of \( px \) are in nondecreasing order (or, more generally, if they are grouped in increasing order of the intervals \( [x(r-1), x(r)] \)). A single call of \( \text{nag\_monotonic\_evaluate} \) with \( m > 1 \) is more efficient than several calls with \( m = 1 \).

10 Example

This example program reads in values of \( n, x, f, d \) and \( m \), and then calls \( \text{nag\_monotonic\_evaluate} \) to evaluate the interpolant at equally spaced points.

10.1 Program Text

```c
#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_evaluate (e01bfc) 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;
    }
    #ifdef _WIN32
        scanf_s("%lf%lf%lf", &x[r], &f[r], &d[r]);
    #else
        scanf("%lf%lf%lf", &x[r], &f[r], &d[r]);
    #endif
    #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);
\}

Mark 25 e01bfc
if (fail.code != NE_NOERROR) {
    printf("Error from nag_monotonic_evaluate (e01bfc)\n\n", fail.message);
    exit_status = 1;
    goto END;
}
printf(" Interpolated
"); printf(" Abscissa Value
"); 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

e01bfc.4 (last) Mark 25

10.2 Program Data

nag_monotonic_evaluate (e01bfc) Example Program Data

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>dy</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.990</td>
<td>0.00000E+0</td>
<td>0.00000E+0</td>
</tr>
<tr>
<td>8.090</td>
<td>0.27643E-4</td>
<td>5.52510E-4</td>
</tr>
<tr>
<td>8.190</td>
<td>0.43749E-1</td>
<td>0.33587E+0</td>
</tr>
<tr>
<td>8.700</td>
<td>0.16918E+0</td>
<td>0.34944E+0</td>
</tr>
<tr>
<td>9.200</td>
<td>0.46943E+0</td>
<td>0.59696E+0</td>
</tr>
<tr>
<td>10.00</td>
<td>0.94374E+0</td>
<td>6.03260E-2</td>
</tr>
<tr>
<td>12.00</td>
<td>0.99864E+0</td>
<td>8.98335E-4</td>
</tr>
<tr>
<td>15.00</td>
<td>0.99992E+0</td>
<td>2.93954E-5</td>
</tr>
<tr>
<td>20.00</td>
<td>0.99999E+0</td>
<td>0.00000E+0</td>
</tr>
</tbody>
</table>

10.3 Program Results

e01bfc.4 (last) Mark 25

nag_monotonic_evaluate (e01bfc) Example Program Results

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.990</td>
<td>0.0000</td>
</tr>
<tr>
<td>9.190</td>
<td>0.4640</td>
</tr>
<tr>
<td>10.392</td>
<td>0.9645</td>
</tr>
<tr>
<td>11.593</td>
<td>0.9965</td>
</tr>
<tr>
<td>12.794</td>
<td>0.9992</td>
</tr>
<tr>
<td>13.995</td>
<td>0.9998</td>
</tr>
<tr>
<td>15.196</td>
<td>0.9999</td>
</tr>
<tr>
<td>16.397</td>
<td>1.0000</td>
</tr>
<tr>
<td>17.598</td>
<td>1.0000</td>
</tr>
<tr>
<td>18.799</td>
<td>1.0000</td>
</tr>
<tr>
<td>20.000</td>
<td>1.0000</td>
</tr>
</tbody>
</table>