nag_monotonic_evaluate (e01bfc)

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

2. **Specification**
   ```c
   #include <nag.h>
   #include <nage01.h>

   void nag_monotonic_evaluate(Integer n, double x[], double f[],
                                double d[], Integer m, 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, ..., 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. **Parameters**
   - **n**
     - Input: n, x, f and d must be unchanged from the previous call of nag_monotonic_interpolant (e01bec).
   - **m**
     - Input: m, the number of points at which the interpolant is to be evaluated.
     - Constraint: m ≥ 1.
   - **px**
     - Input: the m values of x at which the interpolant is to be evaluated.
   - **pf**
     - Output: pf[i] contains the value of the interpolant evaluated at the point px[i], for i = 0, 1, ..., m-1.
   - **fail**
     - The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. **Error Indications and Warnings**
   - **NE_INT_ARG_LT**
     - On entry, n must not be less than 2: n = ⟨value⟩.
     - On entry, m must not be less than 1: m = ⟨value⟩.
   - **NE_NOT_MONOTONIC**
     - On entry, x[r-1] ≥ x[r] for r = ⟨value⟩: x[r-1], x[r] = ⟨values⟩.
     - The values of x[r], for r = 0, 1, ..., n-1, are not in strictly increasing order.
   - **NW_EXTRAPOLATE**
     - Warning – some points in array PX lie outside the range x[0]...x[n-1]. Values at these points are unreliable as they have been computed by extrapolation.

6. **Further Comments**
   The time taken by the function is approximately proportional to the number of evaluation points, m. The evaluation will be most efficient if the elements of px are in non-decreasing order (or, more generally, if they are grouped in increasing order of the intervals [x(r-1),x(r)]). A single call of nag_monotonic_evaluate with m > 1 is more efficient than several calls with m = 1.
6.1. Accuracy

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

6.2. References

Fritsch F N (August 1982) PCHIP Final Specifications Lawrence Livermore National Laboratory report UCID-30194.

7. See Also

nag_monotonic_interpolant (e01bec)

8. Example

This example program reads in values of n, x, f, d and m, and then calls nag_monotonic_evaluate to evaluate the interpolant at equally spaced points.

8.1. Program Text

```c
/* nag_monotonic_evaluate(e01bfc) Example Program */
/* Copyright 1990 Numerical Algorithms Group */
/* Mark 2 revised, 1992. */
*
#include <nag.h>
#include <stdio.h>
#include <nag_stdbib.h>
#include <nage01.h>

#define MMAX 50
#define NMAX 50

main()
{
    Integer i, m, n, r;
    double step, d[NMAX], f[NMAX], pf[MMAX], px[MMAX], x[NMAX];
    static NagError fail;
    fail.print = TRUE;
    Vprintf("e01bfc Example Program Results\n");
    Vscanf("%*\n"); /* Skip to end of line */
    Vscanf("%ld", &n);
    if (n>=1 && n<NMAX)
    {
        for (r=0; r<n; Vscanf("%lf%lf%lf", &x[r], &f[r], &d[r]), r++);
        Vscanf("%ld", &m);
        if (m>=1 && m<MMAX)
        {
            /* 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]);
                e01bfc(n, x, f, d, m, px, pf, &fail);
                Vprintf(" Abscissa Value\n");
                for (i=0; i < m; i++)
                {
                    Vprintf("%13.4f%13.4f\n", px[i], pf[i]);
                }
                exit(EXIT_SUCCESS);
            }
        }
    }
    else
    {
        Vfprintf(stderr, "n is out of range: n = %5ld\n", n);
        exit(EXIT_FAILURE);
    }
}
```
8.2. Program Data

Example Program Data

<table>
<thead>
<tr>
<th>Abscissa Value</th>
<th>Interpolated Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.990</td>
<td>0.00000E+0</td>
</tr>
<tr>
<td>8.090</td>
<td>0.27643E-4</td>
</tr>
<tr>
<td>8.190</td>
<td>0.43749E-1</td>
</tr>
<tr>
<td>8.700</td>
<td>0.16918E+0</td>
</tr>
<tr>
<td>9.200</td>
<td>0.46943E+0</td>
</tr>
<tr>
<td>10.00</td>
<td>0.94374E+0</td>
</tr>
<tr>
<td>12.00</td>
<td>0.99864E+0</td>
</tr>
<tr>
<td>15.00</td>
<td>0.99992E+0</td>
</tr>
<tr>
<td>20.00</td>
<td>0.99999E+0</td>
</tr>
</tbody>
</table>

8.3. Program Results

Example Program Results

<table>
<thead>
<tr>
<th>Abscissa</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.990</td>
<td>0.0000</td>
</tr>
<tr>
<td>9.191</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>