nag_1d_spline_interpolant (e01bac) determines a cubic spline interpolant to a given set of data.

nag_1d_spline_interpolant (e01bac) determines a cubic spline $s(x)$, defined in the range $x_1 \leq x \leq x_m$, which interpolates (passes exactly through) the set of data points $(x_i, y_i)$, for $i = 1, 2, \ldots, m$, where $m \geq 4$ and $x_1 < x_2 < \cdots < x_m$. Unlike some other spline interpolation algorithms, derivative end conditions are not imposed. The spline interpolant chosen has $m - 4$ interior knots $\lambda_5, \lambda_6, \ldots, \lambda_m$, which are set to the values of $x_3, x_4, \ldots, x_{m-2}$ respectively. This spline is represented in its B-spline form (see Cox (1975)):

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

where $N_i(x)$ denotes the normalized B-spline of degree 3, defined upon the knots $\lambda_i, \lambda_{i+1}, \ldots, \lambda_{i+4}$, and $c_i$ denotes its coefficient, whose value is to be determined by the function.

The use of B-splines requires eight additional knots $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_{m+1}, \lambda_{m+2}, \lambda_{m+3}$ and $\lambda_{m+4}$ to be specified; the function sets the first four of these to $x_1$ and the last four to $x_m$.

The algorithm for determining the coefficients is as described in Cox (1975) except that QR factorization is used instead of $LU$ decomposition. The implementation of the algorithm involves setting up appropriate information for the related function nag_1d_spline_fit_knots (e02bac) followed by a call of that function. (For further details of nag_1d_spline_fit_knots (e02bac), see the function document.)

Values of the spline interpolant, or of its derivatives or definite integral, can subsequently be computed as detailed in Section 9.

References


Arguments

1: $m$ – Integer

On entry: $m$, the number of data points.

Constraint: $m \geq 4$. 
2:  \texttt{x[m]} – const double \hspace{1cm} 	extit{Input}

\textit{On entry:} \(x[i-1]\) must be set to \(x_i\), the \(i\)th data value of the independent variable \(x\), for \(i = 1, 2, \ldots, m\).

\textit{Constraint:} \(x[i] < x[i+1]\), for \(i = 0, 1, \ldots, m - 2\).

3:  \texttt{y[m]} – const double \hspace{1cm} 	extit{Input}

\textit{On entry:} \(y[i-1]\) must be set to \(y_i\), the \(i\)th data value of the dependent variable \(y\), for \(i = 1, 2, \ldots, m\).

4:  \texttt{spline} – Nag_Spline *

Pointer to structure of type Nag_Spline with the following members:

\texttt{n} – Integer \hspace{1cm} 	extit{Output}

\textit{On exit:} the size of the storage internally allocated to \texttt{lamda}. This is set to \(m + 4\).

\texttt{lamda} – double * \hspace{1cm} 	extit{Output}

\textit{On exit:} the pointer to which storage of size \texttt{n} is internally allocated. \texttt{lamda}[i-1] contains the \(i\)th knot, for \(i = 1, 2, \ldots, m + 4\).

\texttt{c} – double * \hspace{1cm} 	extit{Output}

\textit{On exit:} the pointer to which storage of size \texttt{n} – 4 is internally allocated. \texttt{c}[i-1] contains the coefficient \(c_i\) of the B-spline \(N_i(x)\), for \(i = 1, 2, \ldots, m\).

Note that when the information contained in the pointers \texttt{lamda} and \texttt{c} is no longer of use, or before a new call to \texttt{nag_1d_spline_interpolant (e01bac)} with the same \texttt{spline}, you should free this storage using the NAG macro \texttt{NAG_FREE}. This storage will not have been allocated if this function returns with \texttt{fail.code} \(
eq\) NE_NOERROR.

5:  \texttt{fail} – NagError * \hspace{1cm} 	extit{Input/Output}

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

6 \hspace{0.5cm} \textbf{Error Indicators and Warnings}

\textbf{NE_ALLOC_FAIL}

Dynamic memory allocation failed.

\textbf{NE_INT_ARG_LT}

On entry, \texttt{m} = \langle value\rangle.

\textit{Constraint:} \texttt{m} \(\geq\) 4.

\textbf{NE_NOT_STRICTLY_INCREASING}

The sequence \texttt{x} is not strictly increasing: \texttt{x[\langle value\rangle]} = \langle value\rangle, \texttt{x[\langle value\rangle]} = \langle value\rangle.

7 \hspace{0.5cm} \textbf{Accuracy}

The rounding errors incurred are such that the computed spline is an exact interpolant for a slightly perturbed set of ordinates \(y_i + \delta y_i\). The ratio of the root-mean-square value of the \(\delta y_i\) to that of the \(y_i\) is no greater than a small multiple of the relative \textit{machine precision}.

8 \hspace{0.5cm} \textbf{Parallelism and Performance}

Not applicable.
9 Further Comments

The time taken by nag_1d_spline_interpolant (e01bac) is approximately proportional to \( m \).

All the \( x_i \) are used as knot positions except \( x_2 \) and \( x_{m-1} \). This choice of knots (see Cox (1977)) means that \( s(x) \) is composed of \( m - 3 \) cubic arcs as follows. If \( m = 4 \), there is just a single arc space spanning the whole interval \( x_1 \) to \( x_4 \). If \( m \geq 5 \), the first and last arcs span the intervals \( x_1 \) to \( x_3 \) and \( x_{m-2} \) to \( x_m \) respectively. Additionally if \( m \geq 6 \), the \( i \)th arc, for \( i = 2, 3, \ldots, m - 4 \), spans the interval \( x_{i+1} \) to \( x_{i+2} \).

After the call

\[
e01bac(m, x, y, \&\text{spline}, \&\text{fail})
\]

the following operations may be carried out on the interpolant \( s(x) \).

The value of \( s(x) \) at \( x = \text{xval} \) can be provided in the variable \( \text{sval} \) by calling the function

\[
e02bbc(xval, \&\text{sval}, \&\text{spline}, \&\text{fail})
\]

The values of \( s(x) \) and its first three derivatives at \( x = \text{xval} \) can be provided in the array \( \text{sdif} \) of dimension 4, by the call

\[
e02bcc(\text{derivs}, xval, \text{sdif}, \&\text{spline}, \&\text{fail})
\]

Here \( \text{derivs} \) must specify whether the left- or right-hand value of the third derivative is required (see nag_1d_spline_deriv (e02bcc) for details). The value of the integral of \( s(x) \) over the range \( x_1 \) to \( x_m \) can be provided in the variable \( \text{sint} \) by

\[
e02bdc(\&\text{spline}, \&\text{sint}, \&\text{fail})
\]

10 Example

The following example program sets up data from 7 values of the exponential function in the interval 0 to 1. nag_1d_spline_interpolant (e01bac) is then called to compute a spline interpolant to these data.

The spline is evaluated by nag_1d_spline_evaluate (e02bbc), at the data points and at points halfway between each adjacent pair of data points, and the spline values and the values of \( e^x \) are printed out.

10.1 Program Text

/* nag_1d_spline_interpolant (e01bac) Example Program.
 * Copyright 2014 Numerical Algorithms Group.
 * Mark 6 revised, 2000.
 */
#include <nag.h>
#include <stdio.h>
#include <math.h>
#include <nag_stdlib.h>
#include <nage01.h>
#include <nage02.h>
#define MMAX 7
int main(void)
{
    Integer exit_status = 0, i, j, m = MMAX;
    NagError fail;
    Nag_Spline spline;
    double fit, *x = 0, xarg, *y = 0;
    INIT_FAIL(fail);
    /* Initialise spline */
    spline.lamda = 0;
spline.c = 0;

printf("nag_1d_spline_interpolant (e01bac) Example Program Results\n");

if (m >= 1)
{
    if (!(y = NAG_ALLOC(m, double)) ||
        !(x = NAG_ALLOC(m, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

} else
{
    exit_status = 1;
    return exit_status;
}

x[0] = 0.0; x[1] = 0.2; x[2] = 0.4;
x[3] = 0.6; x[4] = 0.75; x[5] = 0.9; x[6] = 1.0;

for (i = 0; i < m; ++i)
    y[i] = exp(x[i]);

/* nag_1d_spline_interpolant (e01bac).
   Interpolating function, cubic spline interpolant, one
   variable */

nag_1d_spline_interpolant(m, x, y, &spline, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_1d_spline_interpolant (e01bac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

printf("Number of distinct knots = %"NAG_IFMT"\n", m-2);

for (j = 3; j < m+1; j++)
    printf("%8.4f%s", spline.lamda[j], (j-3)%5 == 4 || j == m?"\n":" ");

for (j = 0; j < m; ++j)
    printf("%"NAG_IFMT" %13.4f\n", j+1, spline.c[j]);

printf("J Abscissa Ordinate Spline\n");

for (j = 0; j < m; ++j)
{
    /* nag_1d_spline_evaluate (e02bbc).
       Evaluation of fitted cubic spline, function only */

    nag_1d_spline_evaluate(x[j], &fit, &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("%"NAG_IFMT" %13.4f %13.4f %13.4f\n", j+1, x[j], y[j], fit);

    xarg = (x[j] + x[j+1]) * 0.5;
    /* nag_1d_spline_evaluate (e02bbc), see above. */
    nag_1d_spline_evaluate(xarg, &fit, &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("%"NAG_IFMT" %13.4f %13.4f %13.4f\n", j+1, xarg, fit);
}
printf("Error from nag_1d_spline_evaluate (e02bbc).\n%s\n",
    fail.message);
exit_status = 1;
goto END;
}  
printf("%13.4f %13.4f
",
    xarg, fit);
}

END:
NAG_FREE(y);
NAG_FREE(x);
NAG_FREE(spline.lamda);
NAG_FREE(spline.c);
return exit_status;

10.2 Program Data
None.

10.3 Program Results
nag_1d_spline_interpolant (e01bac) Example Program Results

Number of distinct knots = 5
Distinct knots located at
0.0000 0.4000 0.6000 0.7500 1.0000

J  B-spline coeff c
1   1.0000
2   1.1336
3   1.3726
4   1.7827
5   2.1744
6   2.4918
7   2.7183

J  Abscissa  Ordinate  Spline
1  0.0000  1.0000  1.0000
    0.1000  1.1052
2  0.2000  1.2214  1.2214
    0.3000  1.3498
3  0.4000  1.4918  1.4918
    0.5000  1.6487
4  0.6000  1.8221  1.8221
    0.6750  1.9640
5  0.7500  2.1170  2.1170
    0.8250  2.2819
6  0.9000  2.4596  2.4596
    0.9500  2.5857
7  1.0000  2.7183  2.7183