NAG C Library Function Document

nag_1d_cheb_interp (e01aec)

1 Purpose

nag_1d_cheb_interp (e01aec) constructs the Chebyshev-series representation of a polynomial interpolant to a set of data which may contain derivative values.

2 Specification

```c
void nag_1d_cheb_interp (Integer m, double xmin, double xmax, const double x[], const double y[], const Integer p[], Integer itmin, Integer itmax, double a[], double perf[], Integer *num_iter, NagError *fail)
```

3 Description

Let \( m \) distinct values \( x_i \) of an independent variable \( x \) be given, with \( x_{\text{min}} \leq x_i \leq x_{\text{max}} \), for \( i = 1, 2, \ldots, m \). For each value \( x_i \), suppose that the value \( y_i \) of the dependent variable \( y \) together with the first \( p_i \) derivatives of \( y \) with respect to \( x \) are given. Each \( p_i \) must therefore be a non-negative integer, with the total number of interpolating conditions, \( n \), equal to \( m + \sum_{i=1}^{m} p_i \).

nag_1d_cheb_interp (e01aec) calculates the unique polynomial \( q(x) \) of degree \( n - 1 \) (or less) which is such that \( q^{(k)}(x_i) = y_i^{(k)} \) for \( i = 1, 2, \ldots, m; k = 0, 1, \ldots, p_i \). Here \( q^{(0)}(x_i) \) means \( q(x_i) \). This polynomial is represented in Chebyshev-series form in the normalised \( \bar{x} \), as follows:

\[
q(x) = \frac{1}{2}a_0 T_0(\bar{x}) + a_1 T_1(\bar{x}) + \ldots + a_{n-1} T_{n-1}(\bar{x}),
\]

where

\[
\bar{x} = \frac{2x - x_{\text{min}} - x_{\text{max}}}{x_{\text{max}} - x_{\text{min}}}
\]

so that \( -1 \leq \bar{x} \leq 1 \) for \( x \) in the interval \( x_{\text{min}} \) to \( x_{\text{max}} \), and where \( T_i(\bar{x}) \) is the Chebyshev polynomial of the first kind of degree \( i \) with argument \( \bar{x} \).

(The polynomial interpolant can subsequently be evaluated for any value of \( x \) in the given range by using nag_1d_cheb_eval2 (e02akc). Chebyshev-series representations of the derivative(s) and integral(s) of \( q(x) \) may be obtained by (repeated) use of nag_1d_cheb_deriv (e02ahc) and nag_1d_cheb_intg (e02ajc).)

The method used consists first of constructing a divided-difference table from the normalised \( \bar{x} \) values and the given values of \( y \) and its derivatives with respect to \( \bar{x} \). The Newton form of \( q(x) \) is then obtained from this table, as described in Huddleston (1974) and Krogh (1970), with the modification described in Section 8.2. The Newton form of the polynomial is then converted to Chebyshev-series form as described in Section 8.3.

Since the errors incurred by these stages can be considerable, a form of iterative refinement is used to improve the solution. This refinement is particularly useful when derivatives of rather high order are given in the data. In reasonable examples, the refinement will usually terminate with a certain accuracy criterion satisfied by the polynomial (see Section 7). In more difficult examples, the criterion may not be satisfied and refinement will continue until the maximum number of iterations (as specified by the input parameter \( \text{itmax} \)) is reached.

In extreme examples, the iterative process may diverge (even though the accuracy criterion is satisfied): if a certain divergence criterion is satisfied, the process terminates at once. In all cases the function returns the ‘best’ polynomial achieved before termination. For the definition of ‘best’ and details of iterative refinement and termination criteria, see Section 8.4.
4 References

Huddleston R E (1974) CDC 6600 routines for the interpolation of data and of data with derivatives SLL-74-0214 Sandia Laboratories (Reprint)


5 Parameters

1: \( m \) – Integer \hspace{1cm} \text{Input}

\text{On entry: } m, \text{ the number of given values of the independent variable } x.

\text{Constraint: } m \geq 1.

2: \( x_{\text{min}} \) – double \hspace{1cm} \text{Input}

3: \( x_{\text{max}} \) – double \hspace{1cm} \text{Input}

\text{On entry: } \text{the lower and upper end-points, respectively, of the interval } [x_{\text{min}}, x_{\text{max}}]. \text{ If they are not determined by the user’s problem, it is recommended that they be set respectively to the smallest and largest values among the } x_i.

\text{Constraint: } x_{\text{min}} < x_{\text{max}}.

4: \( x[m] \) – const double \hspace{1cm} \text{Input}

\text{On entry: } x[i - 1] \text{ must be set to the value of } x_i, \text{ for } i = 1, 2, \ldots, m. \text{ The } x[i] \text{ need not be ordered.}

\text{Constraint: } x_{\text{min}} \leq x[i] \leq x_{\text{max}}, \text{ and the } x[i] \text{ must be distinct.}

5: \( y[\text{dim}] \) – const double \hspace{1cm} \text{Input}

\text{Note: } \text{the dimension, } d, \text{ of the array } y \text{ must be at least } m + \sum_{i=0}^{m-1} p[i].

\text{On entry: } \text{the given values of the dependent variable, and derivatives, as follows:}

The first } p_1 + 1 \text{ elements contain } y_1, y_1^{(1)}, \ldots, y_1^{(p_1)} \text{ in that order.}

The next } p_2 + 1 \text{ elements contain } y_2, y_2^{(1)}, \ldots, y_2^{(p_2)} \text{ in that order.}

\vdots

The last } p_m + 1 \text{ elements contain } y_m, y_m^{(1)}, \ldots, y_m^{(p_m)} \text{ in that order.}

\text{ } m + \sum_{i=0}^{m-1} p[i] \text{ is the total number of interpolating conditions, } n.

6: \( p[m] \) – const Integer \hspace{1cm} \text{Input}

\text{On entry: } p[i - 1] \text{ must be set to } p_i, \text{ the order of the highest-order derivative whose value is given at } x_i, \text{ for } i = 1, 2, \ldots, m. \text{ If the value of } y \text{ only is given for some } x_i \text{ then the corresponding value of } p[i - 1] \text{ must be zero.}

\text{Constraint: } p[i - 1] \geq 0 \text{ for } i = 1, 2, \ldots, m.

7: \( \text{itmin} \) – Integer \hspace{1cm} \text{Input}

8: \( \text{itmax} \) – Integer \hspace{1cm} \text{Input}

\text{On entry: } \text{respectively the minimum and maximum number of iterations to be performed by the function (for full details see Section 8.4, second paragraph). Setting } \text{itmin} \text{ and/or } \text{itmax} \text{ negative or zero invokes default value(s) of 2 and/or 10, respectively.}

The default values will be satisfactory for most problems, but occasionally significant improvement will result from using higher values.

\text{Suggested value: } \text{itmin} = 0 \text{ and } \text{itmax} = 0.
9: \[ a[\text{dim}] \rightarrow \text{double} \]

\text{Output}

\text{Note:} the dimension, \( \text{dim} \), of the array \( a \) must be at least \( m + \sum_{i=0}^{m-1} p[i] \).

\text{On exit:} \( a[i] \) contains the coefficient \( a_i \) in the Chebyshev-series representation of \( q(x) \), for \( i = 0, 1, \ldots, n - 1 \).

10: \[ \text{perf}[\text{dim}] \rightarrow \text{double} \]

\text{Output}

\text{Note:} the dimension, \( \text{dim} \), of the array \( \text{perf} \) must be at least \( \text{ipmax} + m + \sum_{i=0}^{m-1} p[i] + 1 \), where \( \text{ipmax} \) is the maximum element of \( p \).

\text{On exit:} \( \text{perf}[k] \), for \( k = 0, 1, \ldots, \text{ipmax} \), contains the ratio of \( P_k \), the performance index relating to the \( k \)th derivative of the \( q(x) \) finally provided, to 8 times the \textit{machine precision}. \( \text{perf}[\text{ipmax} + j] \), for \( j = 1, 2, \ldots, n \), contains the \( j \)th residual, i.e., the value of \( y_i^{(k)} - q^{(k)}(x_i) \), where \( i \) and \( k \) are the appropriate values corresponding to the \( j \)th element in the array \( y \) (see description of \( y \) in Section 5).

This information is also output if \( \text{fail.code} = \text{NE_ITER_FAIL} \) or \( \text{NE_NOT_ACC} \).

11: \[ \text{num_iter} \rightarrow \text{Integer} \]

\text{Output}

\text{On exit:} \( \text{num_iter} \) contains the number of iterations actually performed in deriving \( q(x) \).

This information is also output if \( \text{fail.code} = \text{NE_ITER_FAIL} \) or \( \text{NE_NOT_ACC} \).

12: \[ \text{fail} \rightarrow \text{NagError} \]

\text{Input/Output}

The NAG error parameter (see the Essential Introduction).

\section{6 Error Indicators and Warnings}

\textbf{NE_INT}

On entry, \( m = \langle \text{value} \rangle \).

Constraint: \( m \geq 1 \).

\textbf{NE_INT_ARRAY}

On entry, \( p[i-1] = \langle \text{value} \rangle \).

Constraint: \( p[i-1] \geq 0 \) for \( i = 1, \ldots, m \).

\textbf{NE_ITER_FAIL}

Iteration is divergent. Problem is ill-conditioned.

\textbf{NE_NOT_ACC}

Not all performance indices are small enough. Try increasing \( \text{imax} \): \( \text{imax} = \langle \text{value} \rangle \).

\textbf{NE_REAL_2}

On entry, \( x_{\text{min}} \geq x_{\text{max}} \): \( x_{\text{min}} = \langle \text{value} \rangle \), \( x_{\text{max}} = \langle \text{value} \rangle \).

\textbf{NE_REAL_ARRAY}

On entry, \( x[i-1] = x[j-1] \): \( x[i-1] = \langle \text{value} \rangle \), \( x[j-1] = \langle \text{value} \rangle \).

\text{On entry,} \( x[i-1] < x_{\text{min}} \) or \( > x_{\text{max}} \): \( x[i-1] = \langle \text{value} \rangle \), \( x_{\text{min}} = \langle \text{value} \rangle \), \( x_{\text{max}} = \langle \text{value} \rangle \).

\textbf{NE_ALLOC_FAIL}

Memory allocation failed.

[\text{NP3645/7}]

\text{e01aec.3}
NE_BAD_PARAM
On entry, parameter (value) had an illegal value.

NE_INTERNAL_ERROR
An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please consult NAG for assistance.

7 Accuracy
A complete error analysis is not currently available, but the method gives good results for reasonable problems.

It is important to realise that for some sets of data, the polynomial interpolation problem is ill-conditioned. That is, a small perturbation in the data may induce large changes in the polynomial, even in exact arithmetic. Though by no means the worst example, interpolation by a single polynomial to a large number of function values given at points equally spaced across the range is notoriously ill-conditioned and the polynomial interpolating such a data set is prone to exhibit enormous oscillations between the data points, especially near the ends of the range. These will be reflected in the Chebyshev coefficients being large compared with the given function values. A more familiar example of ill-conditioning occurs in the solution of certain systems of linear algebraic equations, in which a small change in the elements of the matrix and/or in the components of the right-hand side vector induces a relatively large change in the solution vector. The best that can be achieved in these cases is to make the residual vector small in some sense. If this is possible, the computed solution is exact for a slightly perturbed set of data. Similar considerations apply to the interpolation problem.

The residuals \( y^{(k)}(x_i) - q^{(k)}(x_i) \) are available for inspection. To assess whether these are reasonable, however, it is necessary to relate them to the largest function and derivative values taken by \( q(x) \) over the interval \([x_{\text{min}}, x_{\text{max}}] \). The following performance indices aim to do this. Let the \( k \)th derivative of \( q \) with respect to the normalised variable \( \tilde{x} \) be given by the Chebyshev-series

\[
\frac{1}{2}a_0^k T_0(\tilde{x}) + a_1^k T_1(\tilde{x}) + \ldots + a_{n-1-k}^k T_{n-1-k}(\tilde{x}) .
\]

Let \( A_k \) denote the sum of the moduli of these coefficients (this is an upper bound on the \( k \)th derivative in the interval and is taken as a measure of the maximum size of this derivative), and define

\[
S_k = \max_{r \leq k} A_r .
\]

Then if the root-mean-square value of the residuals of \( q^{(k)} \), scaled so as to relate to the normalised variable \( \tilde{x} \), is denoted by \( r_k \), the performance indices are defined by

\[
P_k = r_k / S_k, \quad \text{for} \quad k = 0, 1, \ldots, \max(p_i) .
\]

It is expected that, in reasonable cases, they will all be less than (say) 8 times the machine precision (this is the accuracy criterion mentioned in Section 3), and in many cases will be of the order of machine precision or less.

8 Further Comments

8.1 Timing
Computation time is approximately proportional to \( IT \times n^3 \), where IT is the number of iterations actually used.

8.2 Divided-difference Strategy
In constructing each new coefficient in the Newton form of the polynomial, a new \( x_i \) must be brought into the computation. The \( x_i \) chosen is that which yields the smallest new coefficient. This strategy increases the stability of the divided-difference technique, sometimes quite markedly, by reducing errors due to cancellation.
8.3 Conversion to Chebyshev Form
Conversion from the Newton form to Chebyshev-series form is effected by evaluating the former at the \( n \) values of \( x \) at which \( T_{n-1}(x) \) takes the value \( \pm 1 \), and then interpolating these \( n \) function values by a call of nag_1d_cheb_interp_fit (e02afc), which provides the Chebyshev-series representation of the polynomial with very small additional relative error.

8.4 Iterative Refinement
The iterative refinement process is performed as follows. First, an initial approximation, \( q_1(x) \) say, is found by the technique described above. The \( r \)th step of the refinement process then consists of evaluating the residuals of the \( r \)th approximation \( q_r(x) \), and constructing an interpolant, \( dq_r(x) \), to these residuals. The next approximation \( q_{r+1}(x) \) to the interpolating polynomial is then obtained as
\[
q_{r+1}(x) = q_r(x) + dq_r(x).
\]
This completes the description of the \( r \)th step.

The iterative process is terminated according to the following criteria. When a polynomial is found whose performance indices (as defined in Section 7) are all less than 8 times the machine precision, the process terminates after \( \text{itmin} \) further iterations (or after a total of \( \text{itmax} \) iterations if that occurs earlier). This will occur in most reasonable problems. The extra iterations are to allow for the possibility of further improvement. If no such polynomial is found, the process terminates after a total of \( \text{itmax} \) iterations. Both these criteria are over-ridden, however, in two special cases. Firstly, if for some value of \( r \) the sum of the moduli of the Chebyshev coefficients of \( dq_r(x) \) is greater than that of \( q_r(x) \), it is concluded that the process is diverging and the process is terminated at once (\( q_{r+1}(x) \) is not computed). Secondly, if at any stage, the performance indices are all computed as zero, again the process is terminated at once.

As the iterations proceed, a record is kept of the best polynomial. Subsequently, at the end of each iteration, the new polynomial replaces the current best polynomial if it satisfies two conditions (otherwise the best polynomial remains unchanged). The first condition is that at least one of its root-mean-square residual values, \( r_k \) (see Section 7) is smaller than the corresponding value for the current best polynomial. The second condition takes two different forms according to whether or not the performance indices (see Section 7) of the current best polynomial are all less than 8 times the machine precision. If they are, then the largest performance index of the new polynomial is required to be less than that of the current best polynomial. If they are not, the number of indices which are less than 8 times machine precision must not be smaller than that of the current best polynomial. When the iterative process is terminated, it is the polynomial then recorded as best, which is returned to the user as \( q(x) \).

9 Example
To construct an interpolant \( q(x) \) to the following data:
\[
m = 4, \quad x_{\text{min}} = 2, \quad x_{\text{max}} = 6, \\
x_1 = 2, \quad p_1 = 0, \quad y_1 = 1, \\
x_2 = 4, \quad p_2 = 1, \quad y_2 = 2, \quad y_2^{(1)} = -1, \\
x_3 = 5, \quad p_3 = 0, \quad y_3 = 1, \\
x_4 = 6, \quad p_4 = 2, \quad y_4 = 2, \quad y_4^{(1)} = 4, \quad y_4^{(2)} = -2.
\]
The coefficients in the Chebyshev-series representation of \( q(x) \) are printed, and also the residuals corresponding to each of the given function and derivative values.

This program is written in a generalised form which can read any number of data-sets.

9.1 Program Text
/* nag_1d_cheb_interp (e01aec) Example Program. */
* * Copyright 2001 Numerical Algorithms Group. * * Mark 7, 2001. */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage01.h>

int main(void)
{
    /* Scalars */
    double xmax, xmin;
    Integer exit_status, i, pmax, ires, iy, j, k, m, n,
             itmin, itmax, num_iter;
    NagError fail;
    /* Arrays */
    double *a = 0, *perf = 0, *x = 0, *y = 0;
    Integer *p = 0;
    exit_status = 0;
    INIT_FAIL(fail);
    Vprintf("e01aec Example Program Results\n");
    /* Skip heading in data file */
    Vscanf("%*[\n] ");
    while (scanf("%ld%lf%lf%*[\n] ", &m, &xmin, &xmax) != EOF)
    {
        if (m > 0)
        {
            /* Allocate memory for p and x. */
            if (!((p = NAG_ALLOC(m, Integer)) ||
                 (x = NAG_ALLOC(m, double)))

                Vprintf("Allocation failure\n");
                exit_status = -1;
                goto END;
            }

            /* Read p, x and y arrays */
            n = 0;
            pmax = 0;
            for (i = 1; i <= m; ++i)
            {
                Vscanf("%ld%lf", &p[i-1], &x[i-1]);
                k = n + p[i-1] + 1;
                /* We need to extend array y as we go along */
                if (!((y = NAG_REALLOC(y, k, double)))

                    Vprintf("Allocation failure\n");
                    exit_status = -1;
                    goto END;
                }

            for (j = n + 1; j <= k; ++j)
                Vscanf("%lf", &y[j-1]);
            Vscanf("%*[\n] ");
            if (p[i-1] > pmax)
                pmax = p[i-1];
            n = k;
        }

        /* Allocate array a */
        if (!((a = NAG_ALLOC(n, double)) ||
             !((perf = NAG_ALLOC(pmax+n+1, double))))

            Vprintf("Allocation failure\n");
            exit_status = -1;
            goto END;
        }

        itmin = -1;
        itmax = -1;
    }

    return exit_status;
}

/* Allocate memory for p and x. */
if (!((p = NAG_ALLOC(m, Integer)) ||
     (x = NAG_ALLOC(m, double)))

    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read p, x and y arrays */
for (i = 1; i <= m; ++i)
{
    Vscanf("%ld%lf")
    k = n + p[i-1] + 1;
    /* We need to extend array y as we go along */
    if (!((y = NAG_REALLOC(y, k, double)))

        Vprintf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    for (j = n + 1; j <= k; ++j)
        Vscanf("%lf")
    Vscanf("%*[\n] ");
    if (p[i-1] > pmax)
        pmax = p[i-1];
    n = k;
}

/* Allocate array a */
if (!((a = NAG_ALLOC(n, double)) ||
     !((perf = NAG_ALLOC(pmax+n+1, double))))

    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

itmin = -1;
itmax = -1;
e01aec(m, xmin, xmax, x, y, p, itmin, itmax, a, perf, &num_iter, &fail);

Vprintf("\n");
if (fail.code == NE_NOERROR)
{
    Vprintf("Total number of interpolating conditions = %ld\n", n);
    Vprintf("\n");
    Vprintf("Interpolating polynomial\n");
    Vprintf("\n");
    Vprintf(" i Chebyshev Coefficient a(i+1)\n");
    for (i = 1; i <= n; ++i)
        Vprintf("%4ld%20.3f\n", i - 1, a[i-1]);
    Vprintf("\n");
    Vprintf(" x R Rth derivative Residual\n");
    iy = 0;
    ires = pmax + 1;
    for (i = 1; i <= m; ++i)
    {
        for (j = 1; j <= p[i-1] + 1; ++j)
        {
            ++iy;
            ++ires;
            if (j - 1 != 0)
                Vprintf(" %4ld%12.1f%17.6f\n", j - 1, y[iy-1], perf[ires-1]);
            else
                Vprintf("%4.1f 0%12.1f%17.6f\n", x[i-1], y[iy-1], perf[ires-1]);
        }
    }
    else
    {
        Vprintf("Error from e01aec.\n%s\n", fail.message);
        exit_status = 1;
    }
}

END:
if (a) NAG_FREE(a);
if (x) NAG_FREE(x);
if (y) NAG_FREE(y);
if (p) NAG_FREE(p);
if (perf) NAG_FREE(perf);
return exit_status;

9.2 Program Data

e01aec Example Program Data

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2.0</td>
<td>6.0</td>
</tr>
<tr>
<td>0</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>4.0</td>
<td>2.0</td>
</tr>
<tr>
<td>0</td>
<td>5.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>6.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>

9.3 Program Results

e01aec Example Program Results

Total number of interpolating conditions = 7
Interpolating polynomial
<table>
<thead>
<tr>
<th>i</th>
<th>Chebyshev Coefficient $a(i+1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9.125</td>
</tr>
<tr>
<td>1</td>
<td>-4.578</td>
</tr>
<tr>
<td>2</td>
<td>0.461</td>
</tr>
<tr>
<td>3</td>
<td>2.852</td>
</tr>
<tr>
<td>4</td>
<td>-2.812</td>
</tr>
<tr>
<td>5</td>
<td>2.227</td>
</tr>
<tr>
<td>6</td>
<td>-0.711</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>R</th>
<th>Rth derivative</th>
<th>Residual</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>0</td>
<td>1.0</td>
<td>0.0000000</td>
</tr>
<tr>
<td>4.0</td>
<td>0</td>
<td>2.0</td>
<td>0.0000000</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>-1.0</td>
<td>-0.0000000</td>
</tr>
<tr>
<td>5.0</td>
<td>0</td>
<td>1.0</td>
<td>-0.0000000</td>
</tr>
<tr>
<td>6.0</td>
<td>0</td>
<td>2.0</td>
<td>-0.0000000</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4.0</td>
<td>0.0000000</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-2.0</td>
<td>0.0000000</td>
</tr>
</tbody>
</table>