NAG 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

```
#include <nag.h>
#include <nage01.h>
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 $x_{\min} \leq x_i \leq x_{\max}$, 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 $i = 1, 2, \ldots, m$ and $k = 0, 1, \ldots, p_i$. Here $q^{(0)}(x_i)$ means $q(x_i)$. This polynomial is represented in Chebyshev series form in the normalized variable $\bar{x}$, as follows:

$$q(x) = \tfrac{1}{2}a_0 T_0(\bar{x}) + a_1 T_1(\bar{x}) + \cdots + a_{n-1} T_{n-1}(\bar{x}),$$

where

$$\bar{x} = \frac{2x - x_{\max} - x_{\min}}{x_{\max} - x_{\min}}$$

so that $-1 \leq \bar{x} \leq 1$ for x in the interval $x_{\min}$ to $x_{\max}$, 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 normalized $\bar{x}$ 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 9.2. The Newton form of the polynomial is then converted to Chebyshev series form as described in Section 9.3.

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 argument **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 9.4.

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)

Krogh F T (1970) Efficient algorithms for polynomial interpolation and numerical differentiation *Math. Comput.* **24** 185–190

5 Arguments

1: **m** – Integer *Input*

On entry: m, the number of given values of the independent variable x.

Constraint: $\mathbf{m} \geq 1$.

2: **xmin** – double *Input*

3: **xmax** – double *Input*

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

Constraint: **xmin** $<$ **xmax**.

4: **x**[**m**] – const double *Input*

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

Constraint: **xmin** $\leq \mathbf{x}[i-1] \leq$ **xmax**, and the $\mathbf{x}[i-1]$ must be distinct.

5: **y**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **y** must be at least $\left(\mathbf{m} + \sum_{i=0}^{\mathbf{m}-1} \mathbf{p}[i]\right)$.

On entry: the given values of the dependent variable, and derivatives, as follows:

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

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

$\vdots$

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

6: **p**[**m**] – const Integer *Input*

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

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

7: **itmin** – Integer *Input*
8: **itmax** – Integer *Input*

On entry: respectively the minimum and maximum number of iterations to be performed by the function (for full details see Section 9.4). Setting **itmin** and/or **itmax** 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.

Suggested value: **itmin** $= 0$ and **itmax** $= 0$.

9: **a**$[dim]$ – double *Output*

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

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: **perf**$[dim]$ – double *Output*

Note: the dimension, *dim*, of the array **perf** must be at least $ipmax + \left(\mathbf{m} + \sum_{i=0}^{\mathbf{m}-1} \mathbf{p}[i]\right) + 1$.

On exit: **perf**$[k-1]$, for $k = 0, 1, \ldots, ipmax$, contains the ratio of P_k, the performance index relating to the kth derivative of the $q(x)$ finally provided, to 8 times the ***machine precision***.

perf$[ipmax + j - 1]$, for $j = 1, 2, \ldots, n$, contains the jth residual, i.e., the value of $y_i^{(k)} - q^{(k)}(x_i)$, where i and k are the appropriate values corresponding to the jth element in the array **y** (see the description of **y** in Section 5).

This information is also output if **fail**.**code** = NE_ITER_FAIL or NE_NOT_ACC.

11: **num_iter** – Integer * *Output*

On exit: **num_iter** contains the number of iterations actually performed in deriving $q(x)$.

This information is also output if **fail**.**code** = NE_ITER_FAIL or NE_NOT_ACC.

12: **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_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 1.

NE_INT_ARRAY

On entry, $\mathbf{p}[\langle value \rangle] = \langle value \rangle$.
Constraint: $\mathbf{p}[i-1] \geq 0$, for $i = 1, 2, \ldots, \mathbf{m}$.

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 contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_ITER_FAIL

Iteration is divergent. Problem is ill-conditioned.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_NOT_ACC

Not all performance indices are small enough. Try increasing **itmax**: **itmax** $= \langle value \rangle$.

NE_REAL_2

On entry, **xmin** $= \langle value \rangle$ and **xmax** $= \langle value \rangle$.
Constraint: **xmin** $<$ **xmax**.

NE_REAL_ARRAY

On entry, $I = \langle value \rangle$, $J = \langle value \rangle$ and $\mathbf{x}[I-1] = \langle value \rangle$.
Constraint: $\mathbf{x}[I-1] \neq \mathbf{x}[J-1]$.

On entry, $I = \langle value \rangle$, $\mathbf{x}[I-1] = \langle value \rangle$, **xmin** $= \langle value \rangle$ and **xmax** $= \langle value \rangle$.
Constraint: **xmin** $\leq \mathbf{x}[I-1] \leq$ **xmax**.

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 dataset 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 $y_i^{(k)} - q^{(k)}(x_i)$ 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 $[x_{\min}, x_{\max}]$. The following performance indices aim to do this. Let the kth derivative of q with respect to the normalized variable $\bar{x}$ be given by the Chebyshev series

$$\tfrac{1}{2}a_0^k T_0(\bar{x}) + a_1^k T_1(\bar{x}) + \cdots + a_{n-1-k}^k T_{n-1-k}(\bar{x}).$$

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

$$S_k = \max_{i \le k} A_i.$$

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

$$P_k = r_k / S_k, \qquad \text{for } k = 0, 1, \ldots, \max_i(p_i).$$

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 Parallelism and Performance

nag_1d_cheb_interp (e01aec) is not threaded in any implementation.

9 Further Comments

9.1 Timing

Computation time is approximately proportional to $it \times n^3$, where it is the number of iterations actually used.

9.2 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.

9.3 Conversion to Chebyshev Form

Conversion from the Newton form to Chebyshev series form is effected by evaluating the former at the n values of $\bar{x}$ at which $T_{n-1}(x)$ takes the value ± 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.

9.4 Iterative Refinement

The iterative refinement process is performed as follows.

Firstly, an initial approximation, $q_1(x)$ say, is found by the technique described in Section 3. The rth step of the refinement process then consists of evaluating the residuals of the rth 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 rth 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 **itmin** further iterations (or after a total of **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 **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 the ***machine precision*** must not be smaller than for the current best polynomial. When the iterative process is terminated, it is the polynomial then recorded as best, which is returned to you as $q(x)$.

10 Example

This example constructs an interpolant $q(x)$ to the following data:

$$\begin{array}{llll} m = 4, & x_{\min} = 2, & x_{\max} = 6, & \\ x_1 = 2, & p_1 = 0, & y_1 = 1, & \\ x_2 = 4, & p_2 = 1, & y_2 = 2, & y_2^{(1)} = -1, \\ x_3 = 5, & p_3 = 0, & y_3 = 1, & \\ x_4 = 6, & p_4 = 2, & y_4 = 2, & y_4^{(1)} = 4, \quad y_4^{(2)} = -2. \end{array}$$

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 generalized form which can read any number of data-sets.

10.1 Program Text

```
/* nag_1d_cheb_interp (e01aec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#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);

  printf("nag_1d_cheb_interp (e01aec) Example Program Results\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
```

```
#else
  scanf("%*[^\n] ");
#endif

#ifdef _WIN32
  while (scanf_s("%" NAG_IFMT "%lf%lf%*[^\n] ", &m, &xmin, &xmax) != EOF)
#else
  while (scanf("%" NAG_IFMT "%lf%lf%*[^\n] ", &m, &xmin, &xmax) != EOF)
#endif
  {
    if (m > 0) {
      /* Allocate memory for p and x. */
      if (!(p = NAG_ALLOC(m, Integer)) || !(x = NAG_ALLOC(m, double)))
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }

      /* Read p, x and y arrays */
      n = 0;
      pmax = 0;
      for (i = 1; i <= m; ++i) {
#ifdef _WIN32
        scanf_s("%" NAG_IFMT "%lf", &p[i - 1], &x[i - 1]);
#else
        scanf("%" NAG_IFMT "%lf", &p[i - 1], &x[i - 1]);
#endif
        k = n + p[i - 1] + 1;
        /* We need to extend array y as we go along */
        if (!(y = NAG_REALLOC(y, k, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
        for (j = n + 1; j <= k; ++j)
#ifdef _WIN32
          scanf_s("%lf", &y[j - 1]);
#else
          scanf("%lf", &y[j - 1]);
#endif
#ifdef _WIN32
        scanf_s("%*[^\n] ");
#else
        scanf("%*[^\n] ");
#endif
        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)))
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }

      itmin = -1;
      itmax = -1;
      /* nag_1d_cheb_interp (e01aec).
       * Interpolating functions, polynomial interpolant, data may
       * include derivative values, one variable
       */
      nag_1d_cheb_interp(m, xmin, xmax, x, y, p, itmin, itmax, a, perf,
                         &num_iter, &fail);

      printf("\n");
```

```
      if (fail.code == NE_NOERROR) {
        printf("Total number of interpolating conditions ="
               " %" NAG_IFMT "\n", n);
        printf("\n");
        printf("Interpolating polynomial\n");
        printf("\n");
        printf("   i    Chebyshev Coefficient a(i+1)\n");

        for (i = 1; i <= n; ++i)
          printf("%4" NAG_IFMT "%20.4f\n", i - 1, a[i - 1]);

        printf("\n");

        printf("  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)
              printf("    %4" NAG_IFMT "%12.1f%17.6f\n",
                     j - 1, y[iy - 1], perf[ires - 1]);
            else
              printf("%4.1f   0%12.1f%17.6f\n",
                     x[i - 1], y[iy - 1], perf[ires - 1]);
          }
        }
      }
      else {
        printf("Error from nag_1d_cheb_interp (e01aec).\n%s\n", fail.message);
        exit_status = 1;
      }
    }
  }

END:
  NAG_FREE(a);
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(p);
  NAG_FREE(perf);

  return exit_status;
}
```

10.2 Program Data

```
nag_1d_cheb_interp (e01aec) Example Program Data
    4       2.0       6.0
    0       2.0       1.0
    1       4.0       2.0      -1.0
    0       5.0       1.0
    2       6.0       2.0       4.0      -2.0
```

10.3 Program Results

```
nag_1d_cheb_interp (e01aec) Example Program Results

Total number of interpolating conditions = 7

Interpolating polynomial

   i    Chebyshev Coefficient a(i+1)
   0              9.1250
   1             -4.5781
   2              0.4609
   3              2.8516
   4             -2.8125
   5              2.2266
```

```
   6             -0.7109

  x     R   Rth derivative    Residual
2.0     0         1.0         0.000000
4.0     0         2.0         0.000000
        1        -1.0        -0.000000
5.0     0         1.0        -0.000000
6.0     0         2.0        -0.000000
        1         4.0         0.000000
        2        -2.0         0.000000
```