Module 8.5: nag_cheb_1d
Chebyshev Series

nag_cheb_1d provides procedures for computing and evaluating Chebyshev polynomial approximation to data sets in one dimension.

Contents

Procedure: nag_cheb_1d_fit

1 Description

nag_cheb_1d_fit computes weighted least-squares polynomial approximations of degrees $0, 1, \ldots, n$, in Chebyshev-series form $a_{i,j}$, for $i = 0, 1, \ldots, n$; $j = 0, 1, \ldots, i$, to the set of data points (x_r, f_r) with weights w_r, for $r = 1, 2, \ldots, m$. The weights are by default set to unity, but you may specify non-default values by supplying the optional argument wt.

The polynomial approximation of degree i is represented as

$$p_i(x) = 0.5a_{i,0}T_0(\bar{x}) + a_{i,1}T_1(\bar{x}) + a_{i,2}T_2(\bar{x}) + \cdots + a_{i,i}T_i(\bar{x}),$$

where $T_j(\bar{x})$ is the Chebyshev polynomial of the first kind of degree j with normalised argument $\bar{x}$. This argument lies in the range -1 to $+1$ and is related to the original variable x by the linear transformation

$$\bar{x} = (2x - (x_{\max} + x_{\min}))/(x_{\max} - x_{\min}).$$

Here $x_{\max}$ and $x_{\min}$ are respectively the largest and smallest values of x_r.

2 Usage

```
USE nag_cheb_1d

CALL nag_cheb_1d_fit(x, f, coeff  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathtt{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m \geq 2$ — the number of data points

$0 \leq n < m$ — the maximum degree required

3.1 Mandatory Arguments

x(m) — real(kind=*wp*), intent(in)

Input: the data points (the independent variables) x_r, for $r = 1, 2, \ldots, m$.

Constraints: the elements of x must be in an increasing order.

f(m) — real(kind=*wp*), intent(in)

Input: the values of the dependent variables f_r, for $r = 1, 2, \ldots, m$.

coeff($0 : n, 0 : n$) — real(kind=*wp*), intent(out)

Output: the coefficients of $T_j(\bar{x})$ in the approximating polynomial of degree i. coeff(i, j) contains the coefficient $a_{i,j}$, for $i = 0, 1, \ldots, n$; $j = 0, 1, \ldots, i$.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

wt(m) — real(kind=wp), intent(in), optional

Input: the values w_r of the weights, for $r = 1, 2, \ldots, m$.

Note: advice on the choice of weights is given in Section 3.3 of the Chapter Introduction.

Default: `wt` = 1.0.

Constraints: `wt`(i) > 0.0, for $i = 1, 2, \ldots, m$.

resid($0 : n$) — real(kind=wp), intent(out), optional

Output: `resid`(i) contains the root mean square residual s_i, for $i = 0, 1, \ldots, n$, as described in Section 6.1. In a satisfactory case, these s_i will decrease steadily as i increases and then settle down to a fairly constant value. If the s_i values settle down in this way, it indicates that the closest polynomial approximation justified by the data has been achieved. The degree which first gives the approximately constant value of s_i the appropriate degree to select. For more detail, see the Further Details section of this module document.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
320	The procedure was unable to allocate enough memory.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The polynomial approximation of degree i is represented as

$$p_i(x) = 0.5a_{i,0}T_0(\bar{x}) + a_{i,1}T_1(\bar{x}) + a_{i,2}T_2(\bar{x}) + \cdots + a_{i,i}T_i(\bar{x}),$$

where $T_j(\bar{x})$ is the Chebyshev polynomial of the first kind of degree j with normalised argument $\bar{x}$.

The approximation of degree i has the property that it minimizes σ_i the sum of squares of the weighted residuals ϵ_r, where

$$\epsilon_r = w_r(f_r - p_i(x_r))$$

and $p_i(x_r)$ is the value of the polynomial approximation of degree i at the rth data point.

For $i = 0, 1, \ldots, n$, the procedure produces the values of $a_{i,j}$, for $j = 0, 1, \ldots, i$, together with the value of the root mean square residual $s_i = \sqrt{\frac{\sigma_i}{m-i-1}}$. In the case $m = i + 1$ the procedure sets the value of s_i to zero. The root-mean-square residual are provided to assist the user in deciding the degree of polynomial which satisfactorily fits the data (for more details see the Further Details section of this module document).

The method employed is due to Forsythe [6] and is based upon the generation of a set of polynomials orthogonal with respect to summation over the normalised data set. The extensions due to Clenshaw [1] to represent these polynomials as well as the approximating polynomials in their Chebyshev-series forms are incorporated. The modifications suggested by Reinsch and Gentleman (see [7]) to the method originally employed by Clenshaw for evaluating the orthogonal polynomials from their Chebyshev-series representations are used to give greater numerical stability.

For further details of the algorithm and its use see [4] and [5].

Subsequent evaluation of the Chebyshev-series representations of the polynomial approximations should be carried out using `nag_cheb_1d_eval`.

The approximating polynomials may exhibit undesirable oscillations (particularly near the ends of the range) if the maximum degree n exceeds a critical value which depends on the number of data points m and their relative positions. As a rough guide, for equally-spaced data, this critical value is about $2\sqrt{m}$. For further details see Hayes [8] page 60.

6.2 Timing

The time taken by the procedure is approximately proportional to $mn(n+10)$.

Procedure: nag_cheb_1d_interp

1 Description

nag_cheb_1d_interp computes the coefficients a_j, for $j = 0, 1, \ldots, n$, in the Chebyshev-series

$$0.5a_0T_0(\bar{x}) + a_1T_1(\bar{x}) + a_2T_2(\bar{x}) + \cdots + a_nT_n(\bar{x}),$$

which interpolates the data f_r at the points

$$\bar{x}_r = \cos(r\pi/n), \quad r = 0, 1, \ldots, n.$$

Here $T_j(\bar{x})$ denotes the Chebyshev polynomial of the first kind of degree j with argument $\bar{x}$. The use of these points minimizes the risk of unwanted fluctuations in the polynomial and is recommended when the data abscissae can be chosen by the user, e.g., when the data is given as a graph. For further advantages of this choice of points, see Clenshaw [3].

In terms of the user's original variables, x say, the values of x at which the data f_r are to be provided are

$$x_r = 0.5(x_{\max} - x_{\min})\cos(r\pi/n) + 0.5(x_{\max} + x_{\min}), \quad r = 0, 1, \ldots, n$$

where $x_{\max}$ and $x_{\min}$ are respectively the upper and lower ends of the range of x over which the user wishes to interpolate.

Truncation of the resulting series after the term involving a_i, say, yields a least-squares approximation to the data. This approximation, $p(\bar{x})$, say, is the polynomial of degree i which minimizes

$$0.5\epsilon_0^2 + \epsilon_1^2 + \epsilon_2^2 + \cdots + \epsilon_{n-1}^2 + 0.5\epsilon_n^2,$$

where the residual $\epsilon_r = p(\bar{x}_r) - f_r$, for $r = 0, 1, \ldots, n$.

2 Usage

```
USE nag_cheb_1d
[value =] nag_cheb_1d_interp(f  [, optional arguments])
```

The function returns an array-valued result of type real(kind=*wp*) and the same SIZE as `f`. nag_cheb_1d_interp$(0:n)$ contains the coefficients of $T_j(\bar{x})$ in the approximating polynomial of degree n. Specifically, nag_cheb_1d_interp(i) contains the coefficient a_i, for $i = 0, 1, \ldots, n$.

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

$n \geq 1$ — the degree of the interpolating polynomial = number of data points − 1

3.1 Mandatory Argument

f$(0:n)$ — real(kind=wp), intent(in)

Input: the values of the function at the special set of points. For $r = 0, 1, \ldots, n$, `f`(r) must contain f_r the value of the dependent variable (ordinate) corresponding to the value $\bar{x}_r = \cos(r\pi/n)$ of the independent variable (abscissa) $\bar{x}$, or equivalently to the value

$$x_r = 0.5(x_{\max} - x_{\min})\cos(r\pi/n) + 0.5(x_{\max} + x_{\min})$$

of the user's original variable x. Here $x_{\max}$ and $x_{\min}$ are respectively the upper and lower ends of the range over which the user wishes to interpolate.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
302	An array argument has an invalid shape.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The method used is based on the application of the three-term recurrence relation due to Clenshaw [1] for the evaluation of the defining expression for the Chebyshev coefficients (see, e.g., Clenshaw [3]). The modifications to this recurrence relation suggested by Reinsch and Gentleman (see [7]) are used to give greater numerical stability.

For further details of the algorithm and its use see [4] and [5].

The algorithm provides the coefficients a_j, for $j = 0, 1, \ldots, n$, in the Chebyshev series form of the polynomial of degree n which interpolates the data. In a satisfactory case, the later coefficients in this series, after some initial significant ones, will exhibit a random behaviour, some positive and some negative, with a size about that of the errors in the data or less. All these 'random' coefficients should be discarded, and the remaining (initial) terms of the series be taken as the approximating polynomial. This truncated polynomial is a least-squares fit to the data, though with the point at each end of the range given half the weight of each of the other points. The following example illustrates a case in which degree 5 or perhaps 6 would be chosen for the approximating polynomial.

j	a_j
0	9.315
1	−8.030
2	0.303
3	−1.483
4	0.256
5	−0.386
6	0.076
7	0.022
8	0.014
9	0.005
10	0.011
11	−0.040
12	0.017
13	−0.054
14	0.010
15	−0.034
16	−0.001

Basically, the value of n used needs to be large enough to exhibit the type of behaviour illustrated in the above example. A value of 16 is suggested as being satisfactory for very many practical problems, the required cosine values for this value of n being given in Cox and Hayes [4], page 11. If a satisfactory fit is not obtained, a spline fit should be tried, or, if the user is prepared to accept a higher degree of polynomial, n should be increased: doubling n is an advantageous strategy, since the set of values $\cos(\pi r/n)$, for $r = 0, 1, \ldots, n$, contains all the values of $\cos(\pi r/2n)$, for $r = 0, 1, \ldots, 2n$, so that the old data set will then be re-used in the new one. Thus, for example, increasing n from 16 to 32 will require only 16 new data points, a smaller number than for any other increase of n. If data points are particularly expensive to obtain, a smaller initial value than 16 may be tried, provided the user is satisfied that the number is adequate to reflect the character of the underlying relationship. Again, the number should be doubled if a satisfactory fit is not obtained.

Subsequent evaluation of the Chebyshev-series representations of the polynomial approximations, perhaps truncated after an appropriate number of terms, should be performed by `nag_cheb_1d_eval`.

6.2 Accuracy

The rounding errors committed are such that the computed coefficients are exact for a slightly perturbed set of ordinates $f_i + \delta f_i$. The ratio of the sum of the absolute values of the δf_i to the sum of the absolute values of the f_i is less than a small multiple of $n\epsilon$, where ϵ is `EPSILON(1.0_`*wp*`)` used in `nag_gen_mat_inv`.

6.3 Timing

The time taken by the procedure is approximately proportional to $n^2 + 30$.

Procedure: nag_cheb_1d_fit_con

1 Description

`nag_cheb_1d_fit_con` computes least-squares polynomial approximations of degrees up to n, for the set of data points (x_r, f_r) with weights w_r, for $r = 1, 2, \ldots, m$. At each of the values `con_x`(r), for $r = 1, 2, \ldots, l$, of the independent variable x, the approximations and their derivatives up to order c_r are constrained to have the user-specified values `con_f`$(r, 0 : c_r)$. If the total number of constraints is given by n_c then the Chebyshev-series coefficients are $a_{i,j}$, for $i = n_c, n_c+1, \ldots, n$; $j = 0, 1, \ldots, i$. The weights are by default set to unity, but you may specify non-default values by supplying the optional argument `wt`.

The polynomial approximation of degree i can be written as

$$p_i(x) = 0.5a_{i,0} + a_{i,1}T_1(\bar{x}) + \cdots + a_{i,j}T_j(\bar{x}) + \cdots + a_{ii}T_i(\bar{x})$$

where $T_j(\bar{x})$ is the Chebyshev polynomial of the first kind of degree j with normalised argument $\bar{x}$. This argument lies in the range -1 to $+1$ and is related to the original variable x by the linear transformation

$$\bar{x} = (2x - (x_{\max} + x_{\min}))/(x_{\max} - x_{\min})$$

where $x_{\min}$ and $x_{\max}$ are respectively the lower and upper end-points of the interval of x over which the polynomials are to be defined.

The polynomial approximation of degree i can be written as

$$p_i(x) = 0.5a_{i,0} + a_{i,1}T_1(\bar{x}) + \cdots + a_{i,j}T_j(\bar{x}) + \cdots + a_{ii}T_i(\bar{x})$$

where $T_j(\bar{x})$ is the Chebyshev polynomial of the first kind of degree j with normalised argument $\bar{x}$.

2 Usage

```
USE nag_cheb_1d

CALL nag_cheb_1d_fit_con(x, f, con_x, con_level, con_f, coeff  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m \geq 1$ — the number of data points
$l \geq 1$ — the number of constrained data points
$d \geq 0$ — the highest-order derivative constrained (= MAXVAL(`con_level`))
n — the maximum degree required

n must satisfy the constraints

$$n_c \leq n \leq n_c + m_d - 1,$$

where m_d is the number of *distinct* values in `x` with non-zero weights and n_c is the total number of constraints ($n_c = l + \text{SUM}($`con_level`$)$).

3.1 Mandatory Arguments

x(m) — real(kind=wp), intent(in)

Input: the data points (the independent variables) x_r, for $r = 1, 2, \ldots, m$.

Constraints: the elements of x must be in non-decreasing order.

f(m) — real(kind=wp), intent(in)

Input: the values of the dependent variables f_r, for $r = 1, 2, \ldots, m$.

con_x(l) — real(kind=wp), intent(in)

Input: con_x(r) must contain the rth value of the independent variable at which a constraint is specified, for $r = 1, 2, \ldots, l$.

Constraints: the elements of con_x need not be ordered but must be distinct and satisfy $x_{\min} \leq$ con_x(r) $\leq x_{\max}$; see the optional arguments x_min and x_max for the definition of $x_{\min}$ and $x_{\max}$.

con_level(l) — integer, intent(in)

Input: con_level(r) must contain c_r, the order of the highest-order derivative specified at con_x(r) for $r = 1, 2, \ldots, l$. $c_r = 0$ implies that the value of the approximation at con_x(r) is specified, but not that of any derivative.

Constraints: con_level(r) ≥ 0, for $r = 1, 2, \ldots, l$.

con_f($l, 0 : d$) — real(kind=wp), intent(in)

Input: the values which the approximating polynomials and their derivatives are required to take at the points specified in con_x. For each value of con_x(r), con_f($r, 0 : c_r$) contains the required value of the approximation, its first derivative, second derivative , … , c_rth derivative, for $r = 1, 2, \ldots, l$.

coeff($0 : n, 0 : n$) — real(kind=wp), intent(out)

Output: the coefficients of $T_j(\bar{x})$ in the approximating polynomial of degree i. coeff(i, j) contains the coefficient $a_{i,j}$, for $i = n_c, n_c + 1, \ldots, n$; $j = 0, 1, \ldots, i$.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

x_min — real(kind=wp), intent(in), optional

x_max — real(kind=wp), intent(in), optional

Input: the lower and upper end-points, respectively, of the interval $[x_{\min}, x_{\max}]$. Unless there are specific reasons to the contrary, it is recommended that x_min and x_max be set respectively to the lowest and highest value among the x and con_x. This avoids the danger of extrapolation provided there is a constraint point or data point with non-zero weight at each end-point.

Default:

x_min = MIN(x_f,MINVAL(con_x)), where x_f is the first value of x with non-zero weight,

x_max = MAX(x_l,MAXVAL(con_x)), where x_l is the last value of x with non-zero weight.

Constraints: x_max > x_min.

wt(m) — real(kind=wp), intent(in), optional

Input: the values w_r of the weights, for $r = 1, 2, \ldots, m$.

Note: advice on the choice of weights is given in Section 3.3 of the Chapter Introduction.

Default: wt = 1.0.

Constraints: wt(r) ≥ 0.0, for $r = 1, 2, \ldots, m$.

resid(0 : n) — real(kind=wp), intent(out), optional

Output: `resid(i)` contains s_i, for $i = n_c, n_c+1, \ldots, n$, the root-mean-square residual corresponding to the approximating polynomial of degree i. In the case where the number of data points with non-zero weight is equal to $n+1-n_c$, s_i is indeterminate: the procedure sets it to zero. In a satisfactory case, these s_i will decrease steadily as i increases and then settle down to a fairly constant value. If the s_i values settle down in this way, it indicates that the closest polynomial approximation justified by the data has been achieved. The degree which first gives the approximately constant value of s_i the appropriate degree to select. For more information, see the Further Details section of this module document.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	The supplied problem is too ill-conditioned. The polynomials $\mu(x)$ and/or $\nu(x)$ cannot be determined. This may occur when the constraint points are very close together, or large in number, or when an attempt is made to constrain high-order derivatives.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 3 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The polynomial approximation of degree i can be written as

$$p_i(x) = 0.5a_{i,0} + a_{i,1}T_1(\bar{x}) + \cdots + a_{i,j}T_j(\bar{x}) + \cdots + a_{i,i}T_i(\bar{x})$$

where $T_j(\bar{x})$ is the Chebyshev polynomial of the first kind of degree j with normalised argument $\bar{x}$.

The approximation of degree i has the property that, subject to the imposed constraints, it minimizes Σ_i, the sum of the squares of the weighted residuals ϵ_r, for $r = 1, 2, \ldots, m$ where

$$\epsilon_r = w_r(f_r - p_i(x_r))$$

and $p_i(x_r)$ is the value of the polynomial approximation of degree i at the rth data point.

For $i = n_c, n_c + 1, \ldots, n$, the procedure produces the values of the coefficients $a_{i,j}$, for $j = 0, 1, \ldots, i$, together with the value of the root mean square residual, s_i, defined as

$$\sqrt{\frac{\sum_i}{(m_d+n_c-i-1)}},$$

where m_d is the number of distinct data points with non-zero weight. The root-mean-square residual is provided to assist the user in deciding the degree of polynomial which satisfactorily fits the data (for more information see Further Details section of this module document).

First the procedure determines a polynomial $\mu(\bar{x})$, of degree $n_c - 1$, which satisfies the given constraints, and a polynomial $\nu(\bar{x})$, of degree n_c, which has value (or derivative) zero wherever a constrained value (or derivative) is specified. It then fits $f_r - \mu(x_r)$, for $r = 1, 2, \ldots, m$ with polynomials of the required degree in $\bar{x}$ each with factor $\nu(\bar{x})$. Finally the coefficients of $\mu(\bar{x})$ are added to the coefficients of these fits to give the coefficients of the constrained polynomial approximations to the data points (x_r, f_r), for $r = 1, 2, \ldots, m$. The method employed is given in Hayes [8]: it is an extension of Forsythe's orthogonal polynomials method (see [6]) as modified by Clenshaw [2].

Values of the approximations may subsequently be computed using `nag_cheb_1d_eval`.

To carry out a least-squares polynomial fit without constraints, use `nag_cheb_1d_fit`.

6.2 Timing

The time taken by the procedure to form the interpolating polynomial is approximately proportional to ${n_c}^3$, and that to form the approximating polynomials is very approximately proportional to $m(n+1)(n+1-n_c)$.

Procedure: nag_cheb_1d_eval

1 Description

nag_cheb_1d_eval evaluates the polynomial

$$0.5a_0T_0(\bar{x}) + a_1T_1(\bar{x}) + a_2T_2(\bar{x}) + \cdots + a_nT_n(\bar{x})$$

for a single value or an array of values. $\bar{x}$ is the normalised value and is related to the original x by the linear transformation

$$\bar{x} = (2x - (x_{\max} + x_{\min}))/(x_{\max} - x_{\min}).$$

x must be in the interval $[x_{\min}, x_{\max}]$.

2 Usage

```
USE nag_cheb_1d
[value =] nag_cheb_1d_eval(a, x  [, optional arguments])
```

The function result is of type real(kind=*wp*). If x is a scalar the result will be a scalar. If x is a rank-1 array the result will be a rank-1 array with the same size as x.

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$n \geq 1$ — the degree of the polynomial

$m \geq 1$ — the number of evaluation points

3.1 Mandatory Arguments

a$(0 : n)$ — real(kind=*wp*), intent(in)

Input: the coefficients of $T_j(\bar{x})$ in the approximating polynomial of degree n. a(i) contains the coefficient a_i, for $i = 0, 1, \ldots, n$.

x(m) **/ x** — real(kind=*wp*), intent(in)

Input: the point(s) x_r, for $r = 1, 2, \ldots, m$, at which the polynomial is to be evaluated.

Note: if $n = 1$, x may be declared as a scalar.

Constraints: x_min $\leq$ x $\leq$ x_max.

x_min — real(kind=*wp*), intent(in)

x_max — real(kind=*wp*), intent(in)

Input: the lower and upper end-points, respectively, of the interval $[x_{\min}, x_{\max}]$.

Constraints: x_max $>$ x_min.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
304	Invalid presence of an optional argument.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

Procedure: nag_cheb_1d_deriv

1 Description

`nag_cheb_1d_deriv` determines the coefficients in the Chebyshev-series representation of the derivative of a polynomial given in Chebyshev-series form.

Given the coefficients a_i, for $i = 0, 1, \ldots, n$, of a polynomial $p(x)$ of degree n, where

$$p(x) = 0.5a_0 + a_1 T_1(\bar{x}) + \cdots + a_n T_n(\bar{x})$$

the procedure returns the coefficients $\bar{a}_i$, for $i = 0, 1, \ldots, n-1$, of the polynomial $q(x)$ of degree $n-1$, where

$$q(x) = \frac{dp(x)}{dx} = 0.5\bar{a}_0 + \bar{a}_1 T_1(\bar{x}) + \cdots + \bar{a}_{n-1} T_{n-1}(\bar{x}).$$

Here $T_j(\bar{x})$ denotes the Chebyshev polynomial of the first kind of degree j with argument $\bar{x}$. It is assumed that the normalised variable $\bar{x}$ in the interval $[-1, +1]$ was obtained from the user's original variable x in the interval $[x_{\min}, x_{\max}]$ by the linear transformation

$$\bar{x} = (2x - (x_{\max} + x_{\min}))/(x_{\max} - x_{\min})$$

and that the user requires the derivative to be with respect to the variable x. If the derivative with respect to $\bar{x}$ is required, set $x_{\max} = 1$ and $x_{\min} = -1$.

2 Usage

```
USE nag_cheb_1d
[value =] nag_cheb_1d_deriv(a, x_min, x_max  [, optional arguments])
```

The function returns an array-valued result of type real(kind=*wp*) with `SIZE(a)` $-$ 1 elements. `nag_cheb_1d_deriv`$(0 : n-1)$ contains the Chebyshev coefficients of the derived polynomial $q(x)$ (the differentiation is with respect to the variable x); see Section 6.1. Specifically, `nag_cheb_1d_deriv`(i) contains the coefficient $\bar{a}_i$, for $i = 0, 1, \ldots, n-1$.

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

$n \geq 0$ — the degree of the polynomial

3.1 Mandatory Arguments

a$(0 : n)$ — real(kind=*wp*), intent(in)

Input: the coefficients of $T_j(\bar{x})$ in the approximating polynomial of degree n. `a`(i) contains the coefficient a_i, for $i = 0, 1, \ldots, n$.

x_min — real(kind=*wp*), intent(in)

x_max — real(kind=*wp*), intent(in)

Input: the lower and upper end-points, respectively, of the interval $[x_{\min}, x_{\max}]$.

Constraints: `x_max` > `x_min`.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The method employed is that of [9], Chapter 8, modified to obtain the derivative with respect to x. Initially setting $\bar{a}_{n+1} = \bar{a}_n = 0$, the procedure forms successively

$$\bar{a}_{i-1} = \bar{a}_{i+1} + \frac{2}{x_{\max} - x_{\min}} 2ia_i, \quad i = n, n-1, \ldots, 1.$$

Values of the derivative can subsequently be computed, from the coefficients obtained, by using `nag_cheb_1d_eval`.

6.2 Accuracy

There is always a loss of precision in numerical differentiation, in this case associated with the multiplication by $2i$ in the above formula for $\bar{a}_{i-1}$.

6.3 Timing

The time taken by the procedure to form the interpolating polynomial is approximately proportional to n.

Procedure: nag_cheb_1d_intg

1 Description

nag_cheb_1d_intg determines the coefficients in the Chebyshev-series representation of the indefinite integral of a polynomial given in Chebyshev-series form.

Given the coefficients a_i, for $i = 0, 1, \ldots, n$, of a polynomial $p(x)$ of degree n, where

$$p(x) = 0.5a_0 + a_1T_1(\bar{x}) + \cdots + a_nT_n(\bar{x}),$$

this procedure returns the coefficients a'_i, for $i = 0, 1, \ldots, n+1$, of the polynomial $q(x)$ of degree $n+1$, where

$$q(x) = 0.5a'_0 + a'_1T_1(\bar{x}) + \cdots + a'_{n+1}T_{n+1}(\bar{x}),$$

and

$$q(x) = \int p(x)\,dx.$$

Here $T_j(\bar{x})$ denotes the Chebyshev polynomial of the first kind of degree j with argument $\bar{x}$. It is assumed that the normalised variable $\bar{x}$ in the interval $[-1, +1]$ was obtained from the user's original variable x in the interval $[x_{\mathrm{min}}, x_{\mathrm{max}}]$ by the linear transformation

$$\bar{x} = (2x - (x_{\mathrm{max}} + x_{\mathrm{min}}))/(x_{\mathrm{max}} - x_{\mathrm{min}})$$

and that the user requires the integral to be with respect to the variable x. If the integral with respect to $\bar{x}$ is required, set $x_{\mathrm{max}} = 1$ and $x_{\mathrm{min}} = -1$.

2 Usage

```
USE nag_cheb_1d

[value =] nag_cheb_1d_intg(a, x_min, x_max  [, optional arguments])
```

The function returns an array-valued result of type real(kind=*wp*) with SIZE(a) + 1 elements. nag_cheb_1d_intg$(0:n+1)$ contains the Chebyshev coefficients of the integral $q(x)$ (the integration is with respect to the variable x); see Section 6.1. Specifically, nag_cheb_1d_intg(i) contains the coefficient a'_i, for $i = 0, 1, \ldots, n+1$.

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

$n \geq 0$ — the degree of the polynomial

3.1 Mandatory Arguments

a$(0:n)$ — real(kind=*wp*), intent(in)

Input: the coefficients of $T_j(\bar{x})$ in the approximating polynomial of degree n. a(i) contains the coefficient a_i, for $i = 0, 1, \ldots, n$.

x_min — real(kind=*wp*), intent(in)

x_max — real(kind=*wp*), intent(in)

Input: the lower and upper end-points, respectively, of the interval $[x_{\min}, x_{\max}]$.

Constraints: `x_max` > `x_min`.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The method employed is that of Chebyshev-series [9], Chapter 8, modified for integrating with respect to x. Initially taking $a_{n+1} = a_{n+2} = 0$, the procedure forms successively

$$a'_i = \frac{a_{i-1} - a_{i+1}}{2i} \times \frac{x_{\max} - x_{\min}}{2}, \quad i = n+1, n, \ldots, 1.$$

The constant coefficient a'_0 is chosen so that $q(x_{\min})$ is equal to zero.

Values of definite integrals can subsequently be computed, from the coefficients obtained, by using `nag_cheb_1d_eval` twice.

6.2 Accuracy

In general there is a gain in precision in numerical integration, in this case associated with the division by $2i$ in the above formula for a'_i.

6.3 Timing

The time taken by the procedure to form the interpolating polynomial is approximately proportional to n.

Example 1: Polynomial Fit, Arbitrary Data Points

Determine weighted least-squares polynomial approximations of degrees $n = 0$, 1, 2, 3 and 4, in Chebyshev-series form, to a set of $m = 11$ prescribed data points.

For the approximation of degrees 4 and 3, determine the Chebyshev-series for first derivatives, second derivatives and indefinite integral. It then tabulates the data and the corresponding values of the approximating polynomial, together with the residual errors, and also the values of the approximating polynomial at points half-way between each pair of adjacent data points. The first and second derivatives are also tabulated for all the points. Finally, evaluate the integral of the polynomials from x_1 to x_{m-1}.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_cheb_1d_ex01

  ! Example Program Text for nag_cheb_1d
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_cheb_1d, ONLY : nag_cheb_1d_eval, nag_cheb_1d_fit, &
   nag_cheb_1d_deriv, nag_cheb_1d_intg
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC ABS, KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j, k, l, m, n
  REAL (wp) :: deriv_1, deriv_2, f_calc, x_max, x_min, x_var
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a_deriv_1(:), a_deriv_2(:), a_intg(:), &
   coeff(:,:), f(:), resid(:), wt(:), x(:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_cheb_1d_ex01'

  READ (nag_std_in,*)          ! Skip heading in data file

  ! Read the size of data
  READ (nag_std_in,*) m

  ! Read the maximum degree of the polynomial
  READ (nag_std_in,*) n

  ALLOCATE (coeff(0:n,0:n),f(m),x(m),wt(m),resid(0:n),a_deriv_1(0:n-1), &
   a_deriv_2(0:n-2),a_intg(0:n+1)) ! Allocate storage

  ! Read in problem data
  DO i = 1, m
    READ (nag_std_in,*) x(i), f(i), wt(i)
  END DO

  ! Determine the fit for degrees up to n
  CALL nag_cheb_1d_fit(x,f,coeff,wt=wt,resid=resid)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) '  Degree    R.M.S.       Chebyshev coeff A(j)'
  WRITE (nag_std_out,*) '           residual      j=0     &
```

```
   &  j=1       j=2       j=3       j=4'
  DO i = 0, n
    WRITE (nag_std_out,'(I6,1p,e14.2,2x,0p,6F10.4)') i, resid(i), &
     (coeff(i,j),j=0,i)
  END DO

  x_min = x(1)
  x_max = x(m)

  l = n
  DO k = 1, 2
    ! Determine coefficients for first derivatives
    a_deriv_1(:l-1) = nag_cheb_1d_deriv(coeff(l,:l),x_min,x_max)

    ! Determine coefficients for second derivatives
    a_deriv_2(:l-2) = nag_cheb_1d_deriv(a_deriv_1(:l-1),x_min,x_max)

    ! Determine coefficients for indefinite integral
    a_intg(:l+1) = nag_cheb_1d_intg(coeff(l,:l),x_min,x_max)

    WRITE (nag_std_out,'(//A,i3)') '  Using degree ', l
    WRITE (nag_std_out,'(A)') '  ================'
    WRITE (nag_std_out,'(/A,6I10)') '  Coefficients of     ', (j,j=0,l+1)
    WRITE (nag_std_out,'(A,I2,A,5F10.4)') ' Polynomial of degree', l, &
     '  ', coeff(l,:l)
    WRITE (nag_std_out,'(A,5F10.4)') ' It''s first derivatives  ', &
     a_deriv_1(:l-1)
    WRITE (nag_std_out,'(A,5F10.4)') ' It''s second derivatives ', &
     a_deriv_2(:l-2)
    WRITE (nag_std_out,'(A,6F10.4)') ' It''s indefinite integral', &
     a_intg(:l+1)

    WRITE (nag_std_out,*)
    WRITE (nag_std_out,'(1x,A,I4)') &
     'Polynomial approximation, residuals and derivatives using degree', l
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) '  Abscissa     Weight   Ordinate &
     & Polynomial  Residual   1st Deriv  2nd Deriv'
    DO i = 1, m
      x_var = x(i)
      f_calc = nag_cheb_1d_eval(coeff(l,:l),x_var,x_min,x_max)
      deriv_1 = nag_cheb_1d_eval(a_deriv_1(:l-1),x_var,x_min,x_max)
      deriv_2 = nag_cheb_1d_eval(a_deriv_2(:l-2),x_var,x_min,x_max)
      WRITE (nag_std_out,'(4f11.4,e11.2,2f11.4)') x_var, wt(i), f(i), &
       f_calc, ABS(f(i)-f_calc), deriv_1, deriv_2
      IF (i==m) EXIT
      x_var = 0.5_wp*(x(i)+x(i+1))
      f_calc = nag_cheb_1d_eval(coeff(l,:l),x_var,x_min,x_max)
      deriv_1 = nag_cheb_1d_eval(a_deriv_1(:l-1),x_var,x_min,x_max)
      deriv_2 = nag_cheb_1d_eval(a_deriv_2(:l-2),x_var,x_min,x_max)
      WRITE (nag_std_out,'(f11.4,22x,f11.4,11x,2f11.4)') x_var, f_calc, &
       deriv_1, deriv_2
    END DO

    WRITE (nag_std_out,*)
    WRITE (nag_std_out,fmt='(1x,A,F3.1,A,F3.1,A,F10.4)') &
     ' Definite Integral (', x(1), ' .. ', x(m-1), ') = ', &
     nag_cheb_1d_eval(a_intg(:l+1),x(m-1),x_min=x_min,x_max=x_max) - &
     nag_cheb_1d_eval(a_intg(:l+1),x(1),x_min=x_min,x_max=x_max)
    l = l - 1
  END DO
```

```
    DEALLOCATE (coeff,f,x,wt,resid,a_deriv_1,a_deriv_2, &
     a_intg)          ! Deallocate storage

  END PROGRAM nag_cheb_1d_ex01
```

2 Program Data

```
Example Program Data for nag_cheb_1d_ex01
 11           : m  (size of data)
  4           : n  (maximum degree of polynomial)
    1.00    10.40     1.00   x(1),f(1),wt(1)
    2.10     7.90     1.00
    3.10     4.70     1.00
    3.90     2.50     1.00
    4.90     1.20     1.00
    5.80     2.20     0.80
    6.50     5.10     0.80
    7.10     9.20     0.70
    7.80    16.10     0.50
    8.40    24.50     0.30
    9.00    35.30     0.20   x(m),f(m),wt(m)
```

3 Program Results

```
 Example Program Results for nag_cheb_1d_ex01

   Degree    R.M.S.        Chebyshev coeff A(j)
            residual       j=0       j=1       j=2       j=3       j=4
     0      4.07E+00     12.1740
     1      4.28E+00     12.2954    0.2740
     2      1.69E+00     20.7345    6.2016    8.1876
     3      6.82E-02     24.1429    9.4065   10.8400    3.0589
     4      4.71E-02     24.0776    9.3202   10.7729    2.9965   -0.0855


  Using degree   4
  ================

  Coefficients of              0         1         2         3         4         5
 Polynomial of degree 4     24.0776    9.3202   10.7729    2.9965   -0.0855
 It's first derivatives      9.1549   10.6018    4.4948   -0.1710
 It's second derivatives     5.0443    4.4948   -0.2566
 It's indefinite integral   51.9846   26.6095    6.3236    7.2389    1.4983   -0.0342

 Polynomial approximation, residuals and derivatives using degree   4

   Abscissa     Weight   Ordinate  Polynomial  Residual   1st Deriv  2nd Deriv
     1.0000     1.0000    10.4000    10.4095   0.95E-02    -1.3586    -2.2292
     1.5500                           9.3631               -2.3776    -1.4797
     2.1000     1.0000     7.9000     7.8687   0.31E-01    -2.9898    -0.7497
     2.6000                           6.3072               -3.2023    -0.1029
     3.1000     1.0000     4.7000     4.7196   0.20E-01    -3.0954     0.5279
     3.5000                           3.5369               -2.7852     1.0210
     3.9000     1.0000     2.5000     2.5174   0.17E-01    -2.2799     1.5039
     4.4000                           1.5902               -1.3800     2.0930
     4.9000     1.0000     1.2000     1.1858   0.14E-01    -0.1896     2.6661
     5.3500                           1.3875                1.1236     3.1681
     5.8000     0.8000     2.2000     2.2305   0.31E-01     2.6598     3.6572
     6.1500                           3.3930                4.0050     4.0286
     6.5000     0.8000     5.1000     5.0490   0.51E-01     5.4788     4.3921
     6.8000                           6.8949                6.8424     4.6975
```

```
   7.1000     0.7000     9.2000     9.1635   0.36E-01     8.2968     4.9971
   7.4500                          12.3805               10.1059     5.3393
   7.8000     0.5000    16.1000    16.2515   0.15E+00    12.0334     5.6737
   8.1000                          20.1210               13.7776     5.9540
   8.4000     0.3000    24.5000    24.5264   0.26E-01    15.6052     6.2286
   8.7000                          29.4923               17.5142     6.4974
   9.0000     0.2000    35.3000    35.0429   0.26E+00    19.5030     6.7604

 Definite Integral (1.0 .. 8.4) =    49.8745


 Using degree   3
 ================

 Coefficients of             0         1         2         3         4
Polynomial of degree 3     24.1429    9.4065   10.8400    3.0589
It's first derivatives      9.2916   10.8400    4.5883
It's second derivatives     5.4200    4.5883
It's indefinite integral   51.9108   26.6058    6.3476    7.2267    1.5294

Polynomial approximation, residuals and derivatives using degree   3

  Abscissa     Weight   Ordinate  Polynomial  Residual   1st Deriv  2nd Deriv
   1.0000     1.0000    10.4000    10.4461   0.46E-01    -1.6059    -1.8783
   1.5500                           9.3106               -2.4655    -1.2474
   2.1000     1.0000     7.9000     7.7977   0.10E+00    -2.9781    -0.6165
   2.6000                           6.2555               -3.1430    -0.0430
   3.1000     1.0000     4.7000     4.7025   0.25E-02    -3.0211     0.5306
   3.5000                           3.5488               -2.7171     0.9894
   3.9000     1.0000     2.5000     2.5533   0.53E-01    -2.2296     1.4482
   4.4000                           1.6435               -1.3621     2.0218
   4.9000     1.0000     1.2000     1.2390   0.39E-01    -0.2078     2.5953
   5.3500                           1.4257                1.0762     3.1115
   5.8000     0.8000     2.2000     2.2425   0.42E-01     2.5925     3.6277
   6.1500                           3.3803                3.9325     4.0292
   6.5000     0.8000     5.1000     5.0116   0.88E-01     5.4129     4.4306
   6.8000                           6.8400                6.7937     4.7748
   7.1000     0.7000     9.2000     9.0982   0.10E+00     8.2778     5.1189
   7.4500                          12.3171               10.1397     5.5204
   7.8000     0.5000    16.1000    16.2123   0.11E+00    12.1420     5.9218
   8.1000                          20.1266               13.9702     6.2660
   8.4000     0.3000    24.5000    24.6048   0.10E+00    15.9016     6.6101
   8.7000                          29.6779               17.9363     6.9542
   9.0000     0.2000    35.3000    35.3769   0.77E-01    20.0741     7.2983

 Definite Integral (1.0 .. 8.4) =    49.7956
```

Example 2: Polynomial Interpolation for Special Data Points

This example is used to determine the Chebyshev coefficients of the polynomial which interpolates the function $f(x) = e^{x/2-0.3}$ in the range $[-1.4, 2.5]$. The function is first evaluated at the data points

$$x_r = 0.5(x_{\max} - x_{\min})\cos(r\pi/n) + 0.5(x_{\max} + x_{\min}), \quad r = 0, 1, \ldots, n$$

using $n = 10$. The procedure `nag_cheb_1d_interp` is used to determine the Chebyshev coefficients.

Evaluate, for comparison with the values of $f(x_r)$, the resulting Chebyshev series at x_r, for $r = 0, 1, \ldots, 10$ using a truncated polynomial (degree 6).

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_cheb_1d_ex02

  ! Example Program Text for nag_cheb_1d
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_cheb_1d, ONLY : nag_cheb_1d_eval, nag_cheb_1d_interp
  USE nag_math_constants, ONLY : nag_pi
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC ABS, COS, EXP, KIND, REAL
  ! .. Parameters ..
  INTEGER, PARAMETER :: n = 10
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j
  REAL (wp) :: pi, x_max, x_min
  ! .. Local Arrays ..
  REAL (wp) :: a(0:n), f(0:n), fit(0:n), x(0:n)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_cheb_1d_ex02'

  pi = nag_pi(0.0_wp)
  x_min = -1.4_wp
  x_max = 2.6_wp

  ! Evaluating
  x = 0.5_wp*(x_max-x_min)*COS((pi/REAL(n,kind=wp))*(/(i,i=0,n)/)) + &
   0.5_wp*(x_max+x_min)

  ! Evaluating  f(x)
  f = EXP(0.5_wp*x-0.3_wp)

  a = nag_cheb_1d_interp(f)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) '        Chebyshev'
  WRITE (nag_std_out,*) '  j   coefficient a(j)'
  WRITE (nag_std_out,'(1X,I3,F14.7)') (j,a(j),j=0,n)

  fit = nag_cheb_1d_eval(a(0:6),x,x_min=x_min,x_max=x_max)
  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) ' Fit using polynomial of degree 6'
  WRITE (nag_std_out,*) '    Abscissa   Ordinate      Fit     Residual'
```

```
    WRITE (nag_std_out,fmt='(1x, 3f11.4,E12.3)') (x(i),f(i),fit(i),ABS(f( &
     i)-fit(i)),i=0,n)

  END PROGRAM nag_cheb_1d_ex02
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_cheb_1d_ex02

         Chebyshev
   j   coefficient a(j)
   0     2.5321318
   1     1.1303182
   2     0.2714953
   3     0.0443368
   4     0.0054742
   5     0.0005429
   6     0.0000450
   7     0.0000032
   8     0.0000002
   9     0.0000000
  10     0.0000000

  Fit using polynomial of degree 6
     Abscissa   Ordinate       Fit     Residual
      2.6000     2.7183     2.7183    0.341E-05
      2.5021     2.5884     2.5884    0.205E-05
      2.2180     2.2457     2.2457    0.917E-06
      1.7756     1.8000     1.8000    0.310E-05
      1.2180     1.3621     1.3621    0.274E-05
      0.6000     1.0000     1.0000    0.199E-06
     -0.0180     0.7342     0.7342    0.242E-05
     -0.5756     0.5556     0.5556    0.297E-05
     -1.0180     0.4453     0.4453    0.104E-05
     -1.3021     0.3863     0.3863    0.173E-05
     -1.4000     0.3679     0.3679    0.301E-05
```

Example 3: Polynomial Fit, Arbitrary Data Points and Constraints

The example program reads data in the following order:

- m, n, l and d

  the number of data points, the maximum degree required, the number of constrained data points and the highest-order derivative constrained respectively

- $(\texttt{x}(i), \texttt{f}(i), i = 1, m)$

  the data points.

- $(\texttt{con\_level}(i), \texttt{con\_x}(i), \texttt{con\_f}(i, 0 : \texttt{con\_level}(i)), i = 1, l)$

  for each variable at which a constraint is specified, $\texttt{con\_x}(i)$, the value of the highest-order derivative specified, $\texttt{con\_level}(i)$, and the values which the approximating polynomials and their derivatives are required to take $\texttt{con\_f}(i, 0 : \texttt{con\_level}(i))$

The program is written in a generalized form which will read any number of data sets.

The data set supplied specifies 5 data points in the interval [0.0,4.0] with unit weights, to which are to be fitted polynomials, p, of degrees up to 4, subject to the 3 constraints:
For the approximation of degree 4, the Chebyshev-series for first derivatives is calculated. It then tabulates the data and the corresponding values of the approximating polynomial, together with the residual errors, and also the values of the first derivatives for the main data and constraints.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_cheb_1d_ex03

  ! Example Program Text for nag_cheb_1d
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_cheb_1d, ONLY : nag_cheb_1d_eval, nag_cheb_1d_fit_con, &
   nag_cheb_1d_deriv
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC ABS, KIND, MAX, MAXVAL, MIN, MINVAL, SUM
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: d, i, l, m, n, np
  REAL (wp) :: deriv, fit, x_max, x_min
  ! .. Local Arrays ..
  INTEGER, ALLOCATABLE :: con_level(:)
  REAL (wp), ALLOCATABLE :: a_deriv(:), coeff(:,:), con_f(:,:), con_x(:), &
   f(:), resid(:), x(:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_cheb_1d_ex03'

  READ (nag_std_in,*)          ! Skip heading in data file

  READ (nag_std_in,*) m, n, l, d
  ALLOCATE (x(m),f(m),con_x(l),con_f(l,0:d),con_level(l),coeff(0:n,0:n), &
   resid(0:n),a_deriv(0:n-1))  ! Allocate storage
```

```
    READ (nag_std_in,*) (x(i),f(i),i=1,m)
    DO i = 1, l
      READ (nag_std_in,*) con_level(i), con_x(i), con_f(i,0:con_level(i))
    END DO

    CALL nag_cheb_1d_fit_con(x,f,con_x,con_level,con_f,coeff,resid=resid)

    x_min = MIN(x(1),MINVAL(con_x))
    x_max = MAX(x(m),MAXVAL(con_x))
    np = l + SUM(con_level)

    a_deriv = nag_cheb_1d_deriv(coeff(n,:),x_min,x_max)

    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) 'Degree  RMS residual'
    WRITE (nag_std_out,'(I5,1PE15.2)') (i,resid(i),i=np,n)
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,'(A,5I10)') '  Coefficients of     ', (i,i=0,n)
    WRITE (nag_std_out,'(A,I2,A,5F10.5)') ' Fit of degree', n, '        ', &
     coeff(n,:)
    WRITE (nag_std_out,'(A,5F10.5)') ' It''s first derivatives ', a_deriv

    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) &
     '  Evaluation at the data points (using fit of degree 4)'
    WRITE (nag_std_out,*) &
     '       x          f         Fit     Residual  1st derive'
    DO i = 1, m

      fit = nag_cheb_1d_eval(coeff(n,:),x(i),x_min,x_max)
      deriv = nag_cheb_1d_eval(a_deriv,x(i),x_min,x_max)

      WRITE (nag_std_out,'(1X,3F11.4,1PE11.2,0PF11.4)') x(i), f(i), fit, &
       ABS(fit-f(i)), deriv
    END DO

    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) &
     '  Evaluation at the constraints (using fit of degree 4)'
    WRITE (nag_std_out,*) &
     '       x          f         Fit     Residual  1st derive  Residual'

    DO i = 1, l

      fit = nag_cheb_1d_eval(coeff(n,:),con_x(i),x_min,x_max)
      deriv = nag_cheb_1d_eval(a_deriv,con_x(i),x_min,x_max)
      IF (con_level(i)==1) THEN

        WRITE (nag_std_out,'(1X,3F11.4,1PE11.2,0PF11.4,1PE11.2)') con_x(i), &
         con_f(i,0), fit, ABS(fit-con_f(i,0)), deriv, ABS(deriv-con_f(i,1))
      ELSE

        WRITE (nag_std_out,'(1X,3F11.4,1PE11.2,0PF11.4)') con_x(i), &
         con_f(i,0), fit, ABS(fit-con_f(i,0)), deriv
      END IF
    END DO

    DEALLOCATE (x,f,con_x,con_f,con_level,coeff,resid, &
     a_deriv)                     ! Deallocate storage

  END PROGRAM nag_cheb_1d_ex03
```

2 Program Data

```
Example Program Data for nag_cheb_1d_ex03
  5  4  2  1   : m, n, l, d
  0.5    0.03
  1.0   -0.75
  2.0   -1.0
  2.5   -0.1
  3.0    1.75  : (x(i),f(i),i=1,m)
   1     0.0   1.0  -2.0 : con_level(1), con_x(1), con_f(1,1:con_level(1)+1)
   0     4.0   9.0       : con_level(2), con_x(2), con_f(2,1:con_level(2)+1)
```

3 Program Results

```
 Example Program Results for nag_cheb_1d_ex03

 Degree  RMS residual
    3       2.55E-03
    4       2.94E-03

  Coefficients of             0         1         2         3         4
 Fit of degree 4          3.99803   3.49954   3.00100   0.50046  -0.00002
 It's first derivatives    5.00092   6.00193   1.50139  -0.00008

   Evaluation at the data points (using fit of degree 4)
        x          f          Fit      Residual  1st derive
      0.5000     0.0300     0.0310   1.02E-03    -1.8134
      1.0000    -0.7500    -0.7508   7.81E-04    -1.2513
      2.0000    -1.0000    -1.0020   2.00E-03     0.9991
      2.5000    -0.1000    -0.0961   3.95E-03     2.6873
      3.0000     1.7500     1.7478   2.17E-03     4.7508

   Evaluation at the constraints (using fit of degree 4)
        x          f          Fit      Residual  1st derive  Residual
      0.0000     1.0000     1.0000   0.00E+00    -2.0000   8.88E-16
      4.0000     9.0000     9.0000   0.00E+00    10.0037
```

Further Details

1 Least-squares Polynomials: Arbitrary Data Points

nag_cheb_1d_fit fits to arbitrary data points, with arbitrary weights, polynomials of all degrees up to a user-supplied maximum degree n. If the user is only seeking a low-degree polynomial, up to degree 5 or 6 say, then $n = 10$ is an appropriate value, providing there are about 20 data points or more. To assist in deciding the degree of polynomial which satisfactorily fits the data, the procedure provides the root-mean-square residual s_i for all degrees $i = 0, 1, \ldots, n$. In a satisfactory case, these s_i will decrease steadily as i increases and then settle down to a fairly constant value, as shown in the example:

i	s_i
0	3.5215
1	0.7708
2	0.1861
3	0.0820
4	0.0554
5	0.0251
6	0.0264
7	0.0280
8	0.0277
9	0.0297
10	0.0271

If the s_i values settle down in this way, it indicates that the closest polynomial approximation justified by the data has been achieved. The degree which first gives the approximately constant value of s_i (degree 5 in the example) is the appropriate degree to select. (Users who are prepared to accept a fit higher than sixth degree should simply find a high enough value of n to enable the type of behaviour indicated by the example to be detected: thus they should seek values of n for which at least 4 or 5 consecutive values of s_i are approximately the same.) If the degree were allowed to go high enough, s_i would, in most cases, eventually start to decrease again, indicating that the data points are being fitted too closely and that undesirable fluctuations are developing between the points. In some cases, particularly with a small number of data points, this final decrease is not distinguishable from the initial decrease in s_i. In such cases, users may seek an acceptable fit by examining the graphs of several of the polynomials obtained. Failing this, they may (a) seek a transformation of variables which improves the behaviour, (b) try fitting a spline, or (c) provide more data points. If data can be provided simply by drawing an approximating curve by hand and reading points from it, use the points discussed in Section 2.

2 Least-squares Polynomials: Selected Data Points

When users are free to choose the x-values of data points, such as when the points are taken from a graph, it is most advantageous when fitting with polynomials to use the values $x_r = \cos(\pi r/n)$, for $r = 0, 1, \ldots, n$ for some value of n, a suitable value for which is discussed at the end of this section. Note that these x_r relate to the variable x after it has been normalised so that its range of interest is -1 to $+1$. nag_cheb_1d_fit may then be used as in Section 1 to seek a satisfactory fit. However, if the ordinate values are of equal weight, as would often be the case when they are read from a graph, nag_cheb_1d_interp is to be preferred, it being simpler to use and faster.

3 Constraints

nag_cheb_1d_fit_con deals with polynomial curves and allows precise values of the fitting function and (if required) all its derivatives up to a given order to be prescribed at one or more values of the independent variable.

4 Evaluation, Differentiation and Integration

`nag_cheb_1d_eval` evaluates polynomial curves. Differentiation and integration of polynomial curves are performed by `nag_cheb_1d_deriv` and `nag_cheb_1d_intg` respectively. The results are provided in Chebyshev-series form and so repeated differentiation and integration are provided. Values of the derivative or integral can then be computed using `nag_cheb_1d_eval`.

References

[1] Clenshaw C W (1955) A note on the summation of Chebyshev series *Math. Tables Aids Comput.* **9** 118–120

[2] Clenshaw C W (1960) Curve fitting with a digital computer *Comput. J.* **2** 170–173

[3] Clenshaw C W (1962) Mathematical tables *Chebyshev Series for Mathematical Functions* HMSO

[4] Cox M G and Hayes J G (1973) Curve fitting: A guide and suite of algorithms for the non-specialist user *NPL Report NAC 26* National Physical Laboratory

[5] Cox M G (1974) A data-fitting package for the non-specialist user *Software for Numerical Mathematics* (ed D J Evans) Academic Press

[6] Forsythe G E (1957) Generation and use of orthogonal polynomials for data fitting with a digital computer *J. Soc. Indust. Appl. Math.* **5** 74–88

[7] Gentleman W M (1969) An error analysis of Goertzel's (Watt's) method for computing Fourier coefficients *Comput. J.* **12** 160–165

[8] Hayes J G (Ed) (1970) Curve fitting by polynomials in one variable *Numerical Approximation to Functions and Data* Athlone Press, London

[9] (1961) Chebyshev-series *Modern Computing Methods, NPL Notes on Applied Science* **16** HMSO (2nd Edition)