# NAG Library Function Document

# nag_1d_cheb_fit_constr (e02agc)

## 1 Purpose

nag_1d_cheb_fit_constr (e02agc) computes constrained weighted least squares polynomial approximations in Chebyshev series form to an arbitrary set of data points. The values of the approximations and any number of their derivatives can be specified at selected points.

## 2 Specification

```
#include <nag.h>
#include <nage02.h>

void nag_1d_cheb_fit_constr (Nag_OrderType order, Integer m, Integer k,
     double xmin, double xmax, const double x[], const double y[],
     const double w[], Integer mf, const double xf[], const double yf[],
     const Integer p[], double a[], double s[], Integer *n, double resid[],
     NagError *fail)
```

## 3 Description

nag_1d_cheb_fit_constr (e02agc) determines least squares polynomial approximations of degrees up to $k$ to the set of data points $(x_r, y_r)$ with weights $w_r$, for $r = 1, 2, \ldots, m$. The value of $k$, the maximum degree required, is to be prescribed by you. At each of the values $xf_r$, for $r = 1, 2, \ldots, mf$, of the independent variable $x$, the approximations and their derivatives up to order $p_r$ are constrained to have one of the values $yf_s$, for $s = 1, 2, \ldots, n$, specified by you, where $n = mf + \sum_{r=0}^{mf} p_r$.

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

$$\epsilon_r = w_r(y_r - f_i(x_r))$$

and $f_i(x_r)$ is the value of the polynomial approximation of degree $i$ at the $r$th data point.

Each polynomial is represented in Chebyshev series form with normalized argument $\bar{x}$. This argument lies in the range $-1$ to $+1$ and is related to the original variable $x$ by the linear transformation

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

where $x_{\min}$ and $x_{\max}$, specified by you, 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

$$\tfrac{1}{2}a_{i,0} + a_{i,1}T_1(\bar{x}) + \cdots + a_{ij}T_j(\bar{x}) + \cdots + a_{ii}T_i(\bar{x})$$

where $T_j(\bar{x})$ is the Chebyshev polynomial of the first kind of degree $j$ with argument $\bar{x}$. For $i = n, n+1, \ldots, k$, the function produces the values of the coefficients $a_{ij}$, for $j = 0, 1, \ldots, i$, together with the value of the root mean square residual,

$$S_i = \sqrt{\frac{\sigma_i}{(m' + n - i - 1)}},$$

where $m'$ is the number of data points with nonzero weight.

Values of the approximations may subsequently be computed using nag_1d_cheb_eval (e02aec) or nag_1d_cheb_eval2 (e02akc).

First nag_1d_cheb_fit_constr (e02agc) determines a polynomial $\mu(\bar{x})$, of degree $n-1$, which satisfies the given constraints, and a polynomial $\nu(\bar{x})$, of degree $n$, which has value (or derivative) zero wherever a constrained value (or derivative) is specified. It then fits $y_r - \mu(x_r)$, 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 $(x_r, y_r)$, for $r = 1, 2, \ldots, m$. The method employed is given in Hayes (1970): it is an extension of Forsythe's orthogonal polynomials method (see Forsythe (1957)) as modified by Clenshaw (see Clenshaw (1960)).

## 4 References

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

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

Hayes J G (ed.) (1970) *Numerical Approximation to Functions and Data* Athlone Press, London

## 5 Arguments

1: **order** – Nag_OrderType *Input*

*On entry*: the **order** argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by **order** = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

*Constraint*: **order** = Nag_RowMajor or Nag_ColMajor.

2: **m** – Integer *Input*

*On entry*: $m$, the number of data points to be fitted.

*Constraint*: **m** $\geq 1$.

3: **k** – Integer *Input*

*On entry*: $k$, the maximum degree required.

*Constraint*: $n \leq$ **k** $\leq m'' + n - 1$ where $n$ is the total number of constraints and $m''$ is the number of data points with nonzero weights and distinct abscissae which do not coincide with any of the **xf**$_r$.

4: **xmin** – double *Input*

5: **xmax** – double *Input*

*On entry*: the lower and upper end points, respectively, of the interval $[x_{\mathrm{min}}, x_{\mathrm{max}}]$. Unless there are specific reasons to the contrary, it is recommended that **xmin** and **xmax** be set respectively to the lowest and highest value among the $x_r$ and $xf_r$. This avoids the danger of extrapolation provided there is a constraint point or data point with nonzero weight at each end point.

*Constraint*: **xmax** > **xmin**.

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

*On entry*: **x**$[r-1]$ must contain the value $x_r$ of the independent variable at the $r$th data point, for $r = 1, 2, \ldots, m$.

*Constraint*: the **x**$[r-1]$ must be in nondecreasing order and satisfy **xmin** $\leq$ **x**$[r-1] \leq$ **xmax**.

7: **y**[**m**] – const double *Input*

*On entry*: **y**$[r-1]$ must contain $y_r$, the value of the dependent variable at the $r$th data point, for $r = 1, 2, \ldots, m$.

8: **w**[**m**] – const double *Input*

*On entry*: $\mathbf{w}[r-1]$ must contain the weight $w_r$ to be applied to the data point $x_r$, for $r = 1,2,\ldots,m$. For advice on the choice of weights see the e02 Chapter Introduction. Negative weights are treated as positive. A zero weight causes the corresponding data point to be ignored. Zero weight should be given to any data point whose $x$ and $y$ values both coincide with those of a constraint (otherwise the denominators involved in the root mean square residuals $S_i$ will be slightly in error).

9: **mf** – Integer *Input*

*On entry*: $mf$, the number of values of the independent variable at which a constraint is specified.

*Constraint*: $\mathbf{mf} \geq 1$.

10: **xf**[**mf**] – const double *Input*

*On entry*: $\mathbf{xf}[r-1]$ must contain $xf_r$, the value of the independent variable at which a constraint is specified, for $r = 1,2,\ldots,\mathbf{mf}$.

*Constraint*: these values need not be ordered but must be distinct and satisfy
$\mathbf{xmin} \leq \mathbf{xf}[r-1] \leq \mathbf{xmax}$.

11: **yf**[*dim*] – const double *Input*

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

*On entry*: the values which the approximating polynomials and their derivatives are required to take at the points specified in **xf**. For each value of $\mathbf{xf}[r-1]$, **yf** contains in successive elements the required value of the approximation, its first derivative, second derivative, $\ldots, p_r$th derivative, for $r = 1,2,\ldots,mf$. Thus the value, $yf_s$, which the $k$th derivative of each approximation ($k = 0$ referring to the approximation itself) is required to take at the point $\mathbf{xf}[r-1]$ must be contained in $\mathbf{yf}[s-1]$, where

$$s = r + k + p_1 + p_2 + \cdots + p_{r-1},$$

where $k = 0,1,\ldots,p_r$ and $r = 1,2,\ldots,mf$. The derivatives are with respect to the independent variable $x$.

12: **p**[**mf**] – const Integer *Input*

*On entry*: $\mathbf{p}[r-1]$ must contain $p_r$, the order of the highest-order derivative specified at $\mathbf{xf}[r-1]$, for $r = 1,2,\ldots,mf$. $p_r = 0$ implies that the value of the approximation at $\mathbf{xf}[r-1]$ is specified, but not that of any derivative.

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

13: **a**[*dim*] – double *Output*

**Note**: the dimension, *dim*, of the array **a** must be at least $(\mathbf{k}+1) \times (\mathbf{k}+1)$.

Where $\mathbf{A}(i,j)$ appears in this document, it refers to the array element

$\mathbf{a}[(j-1) \times (\mathbf{k}+1) + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{a}[(i-1) \times (\mathbf{k}+1) + j - 1]$ when **order** = Nag_RowMajor.

*On exit*: $\mathbf{A}(i+1,j+1)$ contains the coefficient $a_{ij}$ in the approximating polynomial of degree $i$, for $i = n,\ldots,k$ and $j = 0,1,\ldots,i$.

14: **s**[**k** + **1**] – double *Output*

*On exit*: $\mathbf{s}[i]$ contains $S_i$, for $i = n,\ldots,k$, the root mean square residual corresponding to the approximating polynomial of degree $i$. In the case where the number of data points with nonzero weight is equal to $k+1-n$, $S_i$ is indeterminate: the function sets it to zero. For the interpretation of the values of $S_i$ and their use in selecting an appropriate degree, see Section 3.1 in the e02 Chapter Introduction.

15: **n** – Integer * *Output*

*On exit*: contains the total number of constraint conditions imposed: $\mathbf{n} = \mathbf{mf} + p_1 + p_2 + \cdots + p_{\mathbf{mf}}$.

16: **resid**[**m**] – double *Output*

*On exit*: contains weighted residuals of the highest degree of fit determined $(k)$. The residual at $x_r$ is in element **resid**$[r-1]$, for $r = 1, 2, \ldots, m$.

17: **fail** – NagError * *Input/Output*

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

## 6 Error Indicators and Warnings

**NE_ALLOC_FAIL**

Dynamic memory allocation failed.

**NE_BAD_PARAM**

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

**NE_CONSTRAINT**

On entry, **k** = ⟨*value*⟩ and = ⟨*value*⟩.
Constraint: $n \leq \mathbf{k} \leq m'' + n - 1$ where $n$ is the total number of constraints and $m''$ is the number of data points with nonzero weights and distinct abscissae which do not coincide with any of the $\mathbf{xf}_r$.

**NE_ILL_CONDITIONED**

The polynomials $mu(x)$ and/or $nu(x)$ cannot be found. The problem is too ill-conditioned.

**NE_INT**

On entry, **m** = ⟨*value*⟩.
Constraint: $\mathbf{m} \geq 1$.

On entry, **mf** = ⟨*value*⟩.
Constraint: $\mathbf{mf} \geq 1$.

**NE_INT_3**

On entry, $\mathbf{k} + 1 > m'' + \mathbf{n}$, where $m''$ is the number of data points with nonzero weight and distinct abscissae different from all **xf**, and **n** is the total number of constraints: $\mathbf{k} + 1 =$ ⟨*value*⟩, $m'' =$ ⟨*value*⟩ and **n** = ⟨*value*⟩.

**NE_INT_ARRAY**

On entry, **mf** = ⟨*value*⟩ and $\mathbf{p}[r-1] =$ ⟨*value*⟩.
Constraint: $\mathbf{p}[r-1] \geq 0$, for $r = 1, 2, \ldots, \mathbf{mf}$.

**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.

**NE_NOT_MONOTONIC**

On entry, $i = \langle value \rangle$, $\mathbf{x}[i-1] = \langle value \rangle$ and $\mathbf{x}[i-2] = \langle value \rangle$.
Constraint: $\mathbf{x}[i-1] \geq \mathbf{x}[i-2]$.

**NE_REAL_2**

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

**NE_REAL_ARRAY**

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

On entry, $\mathbf{xf}[I-1]$ lies outside interval [**xmin**, **xmax**]: $I = \langle value \rangle$, $\mathbf{xf}[I-1] = \langle value \rangle$, **xmin** $= \langle value \rangle$ and **xmax** $= \langle value \rangle$.

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

On entry, $\mathbf{x}[I-1]$ lies outside interval [**xmin**, **xmax**] for some $I$.

## 7 Accuracy

No complete error analysis exists for either the interpolating algorithm or the approximating algorithm. However, considerable experience with the approximating algorithm shows that it is generally extremely satisfactory. Also the moderate number of constraints, of low-order, which are typical of data fitting applications, are unlikely to cause difficulty with the interpolating function.

## 8 Parallelism and Performance

Not applicable.

## 9 Further Comments

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

To carry out a least squares polynomial fit without constraints, use nag_1d_cheb_fit (e02adc). To carry out polynomial interpolation only, use nag_1d_cheb_interp (e01aec).

## 10 Example

This example reads data in the following order, using the notation of the argument list above:

**mf**

$\mathbf{p}[i-1]$, $\mathbf{xf}[i-1]$, Y-value and derivative values (if any) at $\mathbf{xf}[i-1]$, for $i = 1, 2, \ldots, \mathbf{mf}$

**m**

$\mathbf{x}[i-1]$, $\mathbf{y}[i-1]$, $\mathbf{w}[i-1]$, for $i = 1, 2, \ldots, \mathbf{m}$

**k**, **xmin**, **xmax**

The output is:

the root mean square residual for each degree from $n$ to $k$;

the Chebyshev coefficients for the fit of degree $k$;

the data points, and the fitted values and residuals for the fit of degree $k$.

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

The dataset 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:

$$p(0.0) = 1.0, \quad p'(0.0) = -2.0, \quad p(4.0) = 9.0.$$

## 10.1 Program Text

```
/* nag_1d_cheb_fit_constr (e02agc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage02.h>

int main(void)
{
  /* Scalars */
  double        fiti, xmax, xmin;
  Integer       exit_status, i, iy, j, k, h, m, mf, n, pda, stride;
  NagError      fail;
  Nag_OrderType order;

  /* Arrays */
  double        *a = 0, *s = 0, *w = 0, *resid = 0,
  *x = 0, *xf = 0, *y = 0, *yf = 0;
  Integer       *p = 0;

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I - 1]
  order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J - 1]
  order = Nag_RowMajor;
#endif

  INIT_FAIL(fail);

  exit_status = 0;
  printf("nag_1d_cheb_fit_constr (e02agc) Example Program Results\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  while (scanf("%ld%*[^\n] ", &mf) != EOF)
    {
      if (mf > 0)
        {
          /* Allocate memory for p and xf. */
          if (!(p = NAG_ALLOC(mf, Integer)) ||
              !(xf = NAG_ALLOC(mf, double)))
            {
              printf("Allocation failure\n");
              exit_status = -1;
              goto END;
            }

          /* Read p, xf and yf arrays */
          iy = 1;
          n = mf;
          for (i = 0; i < mf; ++i)
            {
              scanf("%ld%lf", &p[i], &xf[i]);
              h = iy + p[i] + 1;
              /* We need to extend array yf as we go along */
              if (!(yf = NAG_REALLOC(yf, h - 1, double)))
                {
```

```
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
          }
        for (j = iy-1; j < h - 1; ++j)
          scanf("%lf", &yf[j]);
        scanf("%*[^\n] ");
        n += p[i];
        iy = h;
      }
    scanf("%ld%*[^\n] ", &m);

    if (m > 0)
      {
        /* Allocate memory for x, y and w. */
        if (!(x = NAG_ALLOC(m, double)) ||
            !(y = NAG_ALLOC(m, double)) ||
            !(w = NAG_ALLOC(m, double)))
          {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
          }
        for (i = 0; i < m; ++i)
          scanf("%lf%lf%lf", &x[i], &y[i], &w[i]);
        scanf("%*[^\n] ");
        scanf("%ld%lf%lf%*[^\n] ", &k, &xmin, &xmax);
        pda = k + 1;

        /* Allocate arrays a, s and resid */
        if (!(a = NAG_ALLOC((k + 1) * (k + 1), double)) ||
            !(s = NAG_ALLOC((k + 1), double)) ||
            !(resid = NAG_ALLOC(m, double)))
          {
            printf("Allocation failure\n");
            exit_status = -1;
            goto END;
          }

        /* nag_1d_cheb_fit_constr (e02agc).
         * Least-squares polynomial fit, values and derivatives may
         * be constrained, arbitrary data points
         */
        nag_1d_cheb_fit_constr(order, m, k, xmin, xmax, x, y, w, mf, xf,
                               yf, p, a, s, &n, resid, &fail);
        if (fail.code != NE_NOERROR)
          {
            printf(
                    "Error from nag_1d_cheb_fit_constr (e02agc).\n%s\n",
                    fail.message);
            exit_status = 1;
            goto END;
          }

        printf("\n");
        printf("Degree  RMS residual\n");
        for (i = n; i <= k; ++i)
          printf("%4ld%15.2e\n", i, s[i]);
        printf("\n");

        printf("Details of the fit of degree %2ld\n", k);
        printf("\n");
        printf("  Index   Coefficient\n");
        for (i = 0; i < k + 1; ++i)
          printf("%6ld%11.4f\n", i, A(k+1, i+1));
        printf("\n");

        printf(
               "     i      x(i)       y(i)       Fit     Residual\n");
        for (i = 0; i < m; ++i)
          {
                  /* Note that the coefficients of polynomial are stored in the
                   * rows of A hence when the storage is in Nag_ColMajor order
                   * then stride is the first dimension of A, k + 1.
                   * When the storage is in Nag_RowMajor order then stride is 1.
                   */
#ifdef NAG_COLUMN_MAJOR
                  stride = k + 1;
#else
                  stride = 1;
#endif
                  /* nag_1d_cheb_eval2 (e02akc).
                   * Evaluation of fitted polynomial in one variable from
                   * Chebyshev series form
                   */
                  nag_1d_cheb_eval2(k, xmin, xmax, &A(k+1, 1), stride, x[i],
                                    &fiti, &fail);
                  if (fail.code != NE_NOERROR)
                    {
                      printf(
                              "Error from nag_1d_cheb_eval2 (e02akc).\n%s\n",
                              fail.message);
                      exit_status = 1;
                      goto END;
                    }
                  printf("%6ld%11.4f%11.4f%11.4f", i, x[i], y[i],
                          fiti);
                  printf("%11.2e\n", fiti - y[i]);
                }
            }
        }
    }

 END:
  NAG_FREE(a);
  NAG_FREE(s);
  NAG_FREE(w);
  NAG_FREE(resid);
  NAG_FREE(x);
  NAG_FREE(xf);
  NAG_FREE(y);
  NAG_FREE(yf);
  NAG_FREE(p);

  return exit_status;
}
```

## 10.2 Program Data

```
nag_1d_cheb_fit_constr (e02agc) Example Program Data
    2
    1       0.0       1.0      -2.0
    0       4.0       9.0
    5
       0.5      0.03       1.0
       1.0     -0.75       1.0
       2.0     -1.0        1.0
       2.5     -0.1        1.0
       3.0      1.75       1.0
    4       0.0       4.0
```

## 10.3 Program Results

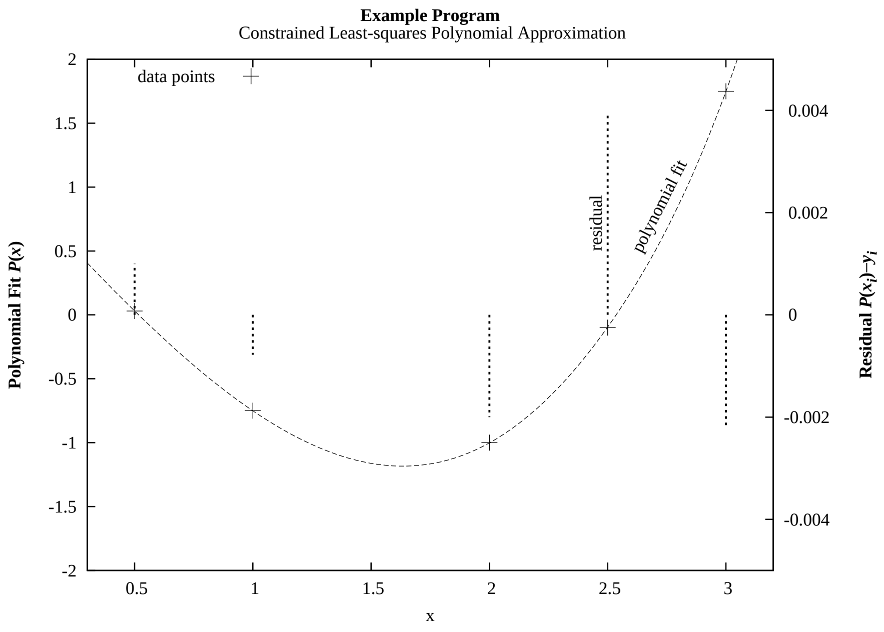
```
nag_1d_cheb_fit_constr (e02agc) Example Program Results

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

Details of the fit of degree  4

  Index   Coefficient
     0      3.9980
     1      3.4995
     2      3.0010
     3      0.5005
     4     -0.0000

     i       x(i)       y(i)        Fit      Residual
     0      0.5000     0.0300     0.0310   1.02e-03
     1      1.0000    -0.7500    -0.7508  -7.81e-04
     2      2.0000    -1.0000    -1.0020  -2.00e-03
     3      2.5000    -0.1000    -0.0961   3.95e-03
     4      3.0000     1.7500     1.7478  -2.17e-03
```


**Example Program**
Constrained Least-squares Polynomial Approximation
data points
residual
polynomial fit
Polynomial Fit $P(x)$
Residual $P(x_i)-y_i$
x