NAG Library Function Document

nag_1d_spline_fit_knots (e02bac)

1 Purpose

nag_1d_spline_fit_knots (e02bac) computes a weighted least squares approximation to an arbitrary set of data points by a cubic spline with knots prescribed by you. Cubic spline interpolation can also be carried out.

2 Specification

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

```
void nag_1d_spline_fit_knots (Integer m, const double x[], const double y[],
     const double weights[], double *ss, Nag_Spline *spline, NagError *fail)
```

3 Description

nag_1d_spline_fit_knots (e02bac) determines a least squares cubic spline approximation $s(x)$ to the set of data points (x_r, y_r) with weights w_r, for $r = 1, 2, \ldots, m$. The value of **spline→n** $= \bar{n} + 7$, where $\bar{n}$ is the number of intervals of the spline (one greater than the number of interior knots), and the values of the knots $\lambda_5, \lambda_6, \ldots, \lambda_{\bar{n}+3}$, interior to the data interval, are prescribed by you.

$s(x)$ has the property that it minimizes θ, the sum of squares of the weighted residuals ϵ_r, for $r = 1, 2, \ldots, m$, where

$$\epsilon_r = w_r(y_{r-s}(x_r)).$$

The function produces this minimizing value of θ and the coefficients $c_1, c_2, \ldots, c_q$, where $q = \bar{n} + 3$, in the B-spline representation

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

Here $N_i(x)$ denotes the normalized B-spline of degree 3 defined upon the knots $\lambda_i, \lambda_{i+1}, \ldots, \lambda_{i+4}$.

In order to define the full set of B-splines required, eight additional knots $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ and $\lambda_{\bar{n}+4}, \lambda_{\bar{n}+5}$, $\lambda_{\bar{n}+6}, \lambda_{\bar{n}+7}$ are inserted automatically by the function. The first four of these are set equal to the smallest x_r and the last four to the largest x_r.

The representation of $s(x)$ in terms of B-splines is the most compact form possible in that only $\bar{n} + 3$ coefficients, in addition to the $\bar{n} + 7$ knots, fully define $s(x)$.

The method employed involves forming and then computing the least squares solution of a set of m linear equations in the coefficients $c_i (i = 1, 2, \ldots, \bar{n} + 3)$. The equations are formed using a recurrence relation for B-splines that is unconditionally stable (Cox (1972), de Boor (1972)), even for multiple (coincident) knots. The least squares solution is also obtained in a stable manner by using orthogonal transformations, viz. a variant of Givens rotations (Gentleman (1974) and Gentleman (1973)). This requires only one equation to be stored at a time. Full advantage is taken of the structure of the equations, there being at most four nonzero values of $N_i(x)$ for any value of x and hence at most four coefficients in each equation.

For further details of the algorithm and its use see Cox (1974), Cox (1975) and Cox and Hayes (1973).

Subsequent evaluation of $s(x)$ from its B-spline representation may be carried out using nag_1d_spline_evaluate (e02bbc). If derivatives of $s(x)$ are also required, nag_1d_spline_deriv (e02bcc) may be used. nag_1d_spline_intg (e02bdc) can be used to compute the definite integral of $s(x)$.

4 References

Cox M G (1972) The numerical evaluation of B-splines *J. Inst. Math. Appl.* **10** 134–149

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

Cox M G (1975) Numerical methods for the interpolation and approximation of data by spline functions *PhD Thesis* City University, London

Cox M G and Hayes J G (1973) Curve fitting: a guide and suite of algorithms for the non-specialist user *NPL Report NAC26* National Physical Laboratory

de Boor C (1972) On calculating with B-splines *J. Approx. Theory* **6** 50–62

Gentleman W M (1973) Least squares computations by Givens transformations without square roots *J. Inst. Math. Applic.* **12** 329–336

Gentleman W M (1974) Algorithm AS 75. Basic procedures for large sparse or weighted linear least squares problems *Appl. Statist.* **23** 448–454

Schoenberg I J and Whitney A (1953) On Polya frequency functions III *Trans. Amer. Math. Soc.* **74** 246–259

5 Arguments

1: **m** – Integer *Input*

On entry: the number m of data points.

Constraint: $\mathbf{m} \geq mdist \geq 4$, where $mdist$ is the number of distinct x values in the data.

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

On entry: the values x_r of the independent variable (abscissa), for $r = 1, 2, \ldots, m$.

Constraint: $x_1 \leq x_2 \leq \ldots \leq x_m$.

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

On entry: the values y_r of the of the dependent variable (ordinate), for $r = 1, 2, \ldots, m$.

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

On entry: the values w_r of the weights, for $r = 1, 2, \ldots, m$. For advice on the choice of weights, see the e02 Chapter Introduction.

Constraint: $w_r > 0$, for $r = 1, 2, \ldots, m$.

5: **ss** – double * *Output*

On exit: the residual sum of squares, θ.

6: **spline** – Nag_Spline *

Pointer to structure of type Nag_Spline with the following members:

n – Integer *Input*

On entry: $\bar{n} + 7$, where $\bar{n}$ is the number of intervals of the spline (which is one greater than the number of interior knots, i.e., the knots strictly within the range x_1 to x_m) over which the spline is defined.

Constraint: $8 \leq \mathbf{n} \leq mdist + 4$, where $mdist$ is the number of distinct x values in the data.

lamda – double * *Input/Output*

On entry: a pointer to which memory of size **n** must be allocated. **lamda**$[i-1]$ must be set to the $(i-4)$th interior knot, λ_i, for $i = 5, 6, \dots, \bar{n} + 3$.

On exit: the input values are unchanged, and **lamda**$[i]$, $i = 0, 1, 2, 3,$**n** $- 4$, **n** $- 3$, **n** $- 2$, **n** $- 1$ contains the additional exterior knots introduced by the function.

Constraint: **x**$[0] <$ **lamda**$[4] \leq$ **lamda**$[5] \leq \dots \leq$ **lamda**$[$**n** $- 5] <$ **x**$[$**m** $- 1]$.

c – double * *Output*

On exit: a pointer to which memory of size **n** $- 4$ is internally allocated. **c** holds the coefficient c_i of the B-spline $N_i(x)$, for $i = 1, 2, \dots, \bar{n} + 3$.

Note that when the information contained in the pointers **lamda** and **c** is no longer of use, or before a new call to nag_1d_spline_fit_knots (e02bac) with the same **spline**, you should free this storage using the NAG macro `NAG_FREE`.

7: **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.

NE_INT_ARG_LT

On entry, **spline→n** must not be less than 8: **spline→n** $= \langle$*value*$\rangle$.

NE_KNOTS_DISTINCT_ABSCI_CONS

Too many knots for the number of distinct abscissae, $mdist$: **spline→n** $= \langle$*value*$\rangle$, $mdist = \langle$*value*$\rangle$.
These must satisfy the constraint **spline→n** $\leq mdist + 4$.

NE_KNOTS_OUTSIDE_DATA_INTVL

On entry, user-specified knots must be interior to the data interval, **spline→lamda**$[4]$ must be greater than **x**$[0]$ and **spline→lamda**$[$**spline→n** $- 5]$ must be less than **x**$[$**m** $- 1]$: **spline→lamda**$[4] = \langle$*value*$\rangle$, **x**$[0] = \langle$*value*$\rangle$, **spline→lamda**$[\langle$*value*$\rangle] = \langle$*value*$\rangle$, **x**$[\langle$*value*$\rangle] = \langle$*value*$\rangle$.

NE_NOT_INCREASING

The sequence **spline→lamda** is not increasing: **spline→lamda**$[\langle$*value*$\rangle] = \langle$*value*$\rangle$, **spline→lamda**$[\langle$*value*$\rangle] = \langle$*value*$\rangle$.
This condition on **spline→lamda** applies to user-specified knots in the interval **spline→lamda**$[4]$, **spline→lamda**$[$**spline→n** $- 5]$.
The sequence **x** is not increasing: **x**$[\langle$*value*$\rangle] = \langle$*value*$\rangle$, **x**$[\langle$*value*$\rangle] = \langle$*value*$\rangle$.

NE_SW_COND_FAIL

The conditions specified by Schoenberg and Whitney fail.
The conditions specified by Schoenberg and Whitney (1953) fail to hold for at least one subset of the distinct data abscissae. That is, there is no subset of **spline→n** $- 4$ strictly increasing values, **x**$[r_0]$, **x**$[r_1], \dots,$ **x**$[r_{\mathbf{spline\to n}-5}]$, among the abscissae such that

x$[r_0] <$ **spline→lamda**$[0] <$ **x**$[r_4]$,

x$[r_1] <$ **spline→lamda**$[1] <$ **x**$[r_5]$,

. . .

$\mathbf{x}[r_{\mathbf{spline}\to\mathbf{n}-9}] <$ **spline→lamda**[**spline→n** $- 9] < \mathbf{x}[r_{\mathbf{spline}\to\mathbf{n}-5}]$.
This means that there is no unique solution: there are regions containing too many knots compared with the number of data points.

NE_WEIGHTS_NOT_POSITIVE

On entry, the weights are not strictly positive: **weights**$[\langle value\rangle] = \langle value\rangle$.

7 Accuracy

The rounding errors committed are such that the computed coefficients are exact for a slightly perturbed set of ordinates $y_r + \delta y_r$. The ratio of the root-mean-square value for the δy_r to the root-mean-square value of the y_r can be expected to be less than a small multiple of $\kappa \times m\times$***machine precision***, where κ is a condition number for the problem. Values of κ for 20-30 practical datasets all proved to lie between 4.5 and 7.8 (see Cox (1975)). (Note that for these datasets, replacing the coincident end knots at the end-points x_1 and x_m used in the function by various choices of non-coincident exterior knots gave values of κ between 16 and 180. Again see Cox (1975) for further details.) In general we would not expect κ to be large unless the choice of knots results in near-violation of the Schoenberg–Whitney conditions.

A cubic spline which adequately fits the data and is free from spurious oscillations is more likely to be obtained if the knots are chosen to be grouped more closely in regions where the function (underlying the data) or its derivatives change more rapidly than elsewhere.

8 Parallelism and Performance

nag_1d_spline_fit_knots (e02bac) is not threaded in any implementation.

9 Further Comments

The time taken by nag_1d_spline_fit_knots (e02bac) is approximately $C \times (2m + \bar{n} + 7)$ seconds, where C is a machine-dependent constant.

Multiple knots are permitted as long as their multiplicity does not exceed 4, i.e., the complete set of knots must satisfy $\lambda_i < \lambda_{i+4}$, for $i = 1, 2, \ldots, \bar{n} + 3$, (see Section 6). At a knot of multiplicity one (the usual case), $s(x)$ and its first two derivatives are continuous. At a knot of multiplicity two, $s(x)$ and its first derivative are continuous. At a knot of multiplicity three, $s(x)$ is continuous, and at a knot of multiplicity four, $s(x)$ is generally discontinuous.

The function can be used efficiently for cubic spline interpolation, i.e., if $m = \bar{n} + 3$. The abscissae must then of course satisfy $x_1 < x_2 < \cdots < x_m$. Recommended values for the knots in this case are $\lambda_i = x_{i-2}$, for $i = 5, 6, \ldots, \bar{n} + 3$.

10 Example

Determine a weighted least squares cubic spline approximation with five intervals (four interior knots) to a set of 14 given data points. Tabulate the data and the corresponding values of the approximating spline, together with the residual errors, and also the values of the approximating spline at points half-way between each pair of adjacent data points.

The example program is written in a general form that will enable a cubic spline approximation with $\bar{n}$ intervals ($\bar{n} - 1$ interior knots) to be obtained to m data points, with arbitrary positive weights, and the approximation to be tabulated. Note that nag_1d_spline_evaluate (e02bbc) is used to evaluate the approximating spline. The program is self-starting in that any number of datasets can be supplied.

10.1 Program Text

```
/* nag_1d_spline_fit_knots (e02bac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 *
 */

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

int main(void)
{
  Integer exit_status = 0, j, m, ncap, ncap7, r, wght;
  NagError fail;
  Nag_Spline spline;
  double fit, ss, *weights = 0, *x = 0, xarg, *y = 0;

  INIT_FAIL(fail);

  /* Initialize spline */
  spline.lamda = 0;
  spline.c = 0;

  printf("nag_1d_spline_fit_knots (e02bac) Example Program Results\n");
#ifdef _WIN32
  scanf_s("%*[^\n]"); /* Skip heading in data file */
#else
  scanf("%*[^\n]"); /* Skip heading in data file */
#endif
#ifdef _WIN32
  while (scanf_s("%" NAG_IFMT "", &m) != EOF)
#else
  while (scanf("%" NAG_IFMT "", &m) != EOF)
#endif
  {
    if (m >= 4) {
      if (!(weights = NAG_ALLOC(m, double)) ||
          !(x = NAG_ALLOC(m, double)) || !(y = NAG_ALLOC(m, double))
             )
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }
    }
    else {
      printf("Invalid m.\n");
      exit_status = 1;
      goto END;
    }
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "", &ncap, &wght);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "", &ncap, &wght);
#endif
    if (ncap > 0) {
      ncap7 = ncap + 7;
      spline.n = ncap7;
      if (!(spline.lamda = NAG_ALLOC(ncap7, double)))
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }
    }
    else {
      printf("Invalid ncap.\n");
      exit_status = 1;
      goto END;
    }
    for (j = 4; j < ncap + 3; ++j)
#ifdef _WIN32
      scanf_s("%lf", &(spline.lamda[j]));
#else
      scanf("%lf", &(spline.lamda[j]));
#endif
    for (r = 0; r < m; ++r) {
      if (wght == 1) {
#ifdef _WIN32
        scanf_s("%lf%lf", &x[r], &y[r]);
#else
        scanf("%lf%lf", &x[r], &y[r]);
#endif
        weights[r] = 1.0;
      }
      else
#ifdef _WIN32
        scanf_s("%lf%lf%lf", &x[r], &y[r], &weights[r]);
#else
        scanf("%lf%lf%lf", &x[r], &y[r], &weights[r]);
#endif
    }
    /* nag_1d_spline_fit_knots (e02bac).
     * Least squares curve cubic spline fit (including
     * interpolation), one variable
     */
    nag_1d_spline_fit_knots(m, x, y, weights, &ss, &spline, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_1d_spline_fit_knots (e02bac).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

    printf("\nNumber of distinct knots = %" NAG_IFMT "\n\n", ncap + 1);
    printf("Distinct knots located at \n\n");
    for (j = 3; j < ncap + 4; j++)
      printf("%8.4f%s", spline.lamda[j],
             (j - 3) % 6 == 5 || j == ncap + 3 ? "\n" : " ");
    printf("\n\n    J    B-spline coeff c\n\n");
    for (j = 0; j < ncap + 3; ++j)
      printf("    %" NAG_IFMT "  %13.4f\n", j + 1, spline.c[j]);
    printf("\nResidual sum of squares = ");
    printf("%11.2e\n\n", ss);
    printf("Cubic spline approximation and residuals\n");
    printf("  r        Abscissa        Weight       Ordinate"
           "          Spline      Residual\n\n");
    for (r = 0; r < m; ++r) {
      /* nag_1d_spline_evaluate (e02bbc).
       * Evaluation of fitted cubic spline, function only
       */
      nag_1d_spline_evaluate(x[r], &fit, &spline, &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_1d_spline_evaluate (e02bbc).\n%s\n",
               fail.message);
        exit_status = 1;
        goto END;
      }

      printf("%3" NAG_IFMT "    %11.4f    %11.4f    %11.4f     %11.4f"
             "    %10.1e\n", r + 1, x[r], weights[r], y[r], fit, fit - y[r]);
      if (r < m - 1) {
        xarg = (x[r] + x[r + 1]) * 0.5;
        /* nag_1d_spline_evaluate (e02bbc), see above. */
        nag_1d_spline_evaluate(xarg, &fit, &spline, &fail);
        if (fail.code != NE_NOERROR) {
          printf("Error from nag_1d_spline_evaluate (e02bbc).\n%s\n",
                 fail.message);
          exit_status = 1;
          goto END;
        }
        printf("    %14.4f             %33.4f\n", xarg, fit);
      }
    }
    /* Free memory used by spline */
    NAG_FREE(spline.lamda);
    NAG_FREE(spline.c);
  END:
    NAG_FREE(weights);
    NAG_FREE(x);
    NAG_FREE(y);
  }
  return exit_status;
}
```

10.2 Program Data

```
nag_1d_spline_fit_knots (e02bac) Example Program Data
  14
   5   2
     1.50
     2.60
     4.00
     8.00
     0.20     0.00     0.20
     0.47     2.00     0.20
     0.74     4.00     0.30
     1.09     6.00     0.70
     1.60     8.00     0.90
     1.90     8.62     1.00
     2.60     9.10     1.00
     3.10     8.90     1.00
     4.00     8.15     0.80
     5.15     7.00     0.50
     6.17     6.00     0.70
     8.00     4.54     1.00
    10.00     3.39     1.00
    12.00     2.56     1.00
```

10.3 Program Results

```
nag_1d_spline_fit_knots (e02bac) Example Program Results

Number of distinct knots = 6

Distinct knots located at

  0.2000   1.5000   2.6000   4.0000   8.0000  12.0000


    J    B-spline coeff c

    1        -0.0465
    2         3.6150
    3         8.5724
    4         9.4261
    5         7.2716
    6         4.1207
    7         3.0822
    8         2.5597

Residual sum of squares =    1.78e-03

Cubic spline approximation and residuals
  r        Abscissa        Weight       Ordinate          Spline      Residual
  1       0.2000        0.2000        0.0000       -0.0465     -4.7e-02
          0.3350                                    1.0622
  2       0.4700        0.2000        2.0000        2.1057      1.1e-01
          0.6050                                    3.0817
  3       0.7400        0.3000        4.0000        3.9880     -1.2e-02
          0.9150                                    5.0558
  4       1.0900        0.7000        6.0000        5.9983     -1.7e-03
          1.3450                                    7.1376
  5       1.6000        0.9000        8.0000        7.9872     -1.3e-02
          1.7500                                    8.3544
  6       1.9000        1.0000        8.6200        8.6348      1.5e-02
          2.2500                                    9.0076
  7       2.6000        1.0000        9.1000        9.0896     -1.0e-02
          2.8500                                    9.0353
  8       3.1000        1.0000        8.9000        8.9125      1.2e-02
          3.5500                                    8.5660
  9       4.0000        0.8000        8.1500        8.1321     -1.8e-02
          4.5750                                    7.5592
 10       5.1500        0.5000        7.0000        6.9925     -7.5e-03
          5.6600                                    6.5010
 11       6.1700        0.7000        6.0000        6.0255      2.6e-02
          7.0850                                    5.2292
 12       8.0000        1.0000        4.5400        4.5315     -8.5e-03
          9.0000                                    3.9045
 13      10.0000        1.0000        3.3900        3.3928      2.8e-03
         11.0000                                    2.9574
 14      12.0000        1.0000        2.5600        2.5597     -3.5e-04
```