# nag_1d_spline_fit (e02bec)

## 1. Purpose

**nag_1d_spline_fit (e02bec)** computes a cubic spline approximation to an arbitrary set of data points. The knots of the spline are located automatically, but a single parameter must be specified to control the trade-off between closeness of fit and smoothness of fit.

## 2. Specification

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

void nag_1d_spline_fit(Nag_Start start, Integer m, double x[], double y[],
            double weights[], double s, Integer nest, double *fp,
            Nag_Comm *warmstartinf, Nag_Spline *spline,
            NagError *fail)
```

## 3. Description

This function determines a smooth 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 spline is given in the B-spline representation

$$s(x) = \sum_{i=1}^{n-4} c_i N_i(x), \tag{1}$$

where $N_i(x)$ denotes the normalised cubic B-spline defined upon the knots $\lambda_i, \lambda_{i+1}, \ldots, \lambda_{i+4}$.

The total number $n$ of these knots and their values $\lambda_1, \ldots, \lambda_n$ are chosen automatically by the function. The knots $\lambda_5, \ldots, \lambda_{n-4}$ are the interior knots; they divide the approximation interval $[x_1, x_m]$ into $n-7$ sub-intervals. The coefficients $c_1, c_2, \ldots, c_{n-4}$ are then determined as the solution of the following constrained minimization problem:

minimize

$$\eta = \sum_{i=5}^{n-4} \delta_i^2 \tag{2}$$

subject to the constraint

$$\theta = \sum_{r=1}^{m} \varepsilon_r^2 \le S, \tag{3}$$

where

$\delta_i$ stands for the discontinuity jump in the third order derivative of $s(x)$ at the interior knot $\lambda_i$,
$\varepsilon_r$ denotes the weighted residual $w_r(y_r - s(x_r))$, and
$S$ is a non-negative number to be specified by the user.

The quantity $\eta$ can be seen as a measure of the (lack of) smoothness of $s(x)$, while closeness of fit is measured through $\theta$. By means of the parameter $S$, 'the smoothing factor', the user will then control the balance between these two (usually conflicting) properties. If $S$ is too large, the spline will be too smooth and signal will be lost (underfit); if $S$ is too small, the spline will pick up too much noise (overfit). In the extreme cases the function will return an interpolating spline ($\theta = 0$) if $S$ is set to zero, and the weighted least-squares cubic polynomial ($\eta = 0$) if $S$ is set very large. Experimenting with $S$ values between these two extremes should result in a good compromise. (See Section 6.3 for advice on choice of $S$.)

The method employed is outlined in Section 6.4 and fully described in Dierckx (1975), Dierckx (1981) and Dierckx (1982). It involves an adaptive strategy for locating the knots of the cubic spline (depending on the function underlying the data and on the value of $S$), and an iterative method for solving the constrained minimization problem once the knots have been determined.

Values of the computed spline, or of its derivatives or definite integral, can subsequently be computed by calling nag_1d_spline_evaluate (e02bbc), nag_1d_spline_deriv (e02bcc) or nag_1d_spline_intg (e02bdc), as described in Section 6.5 .

## 4. Parameters

**start**

Input: **start** must be set to **Nag_Cold** or **Nag_Warm**.

If **start** = **Nag_Cold** (cold start), the function will build up the knot set starting with no interior knots. No values need be assigned to the parameter **spline.n**, and memory will be allocated internally to **spline.lamda**, **spline.c**, **warmstartinf.nag_w** and **warmstartinf.nag_iw**.

If **start** = **Nag_Warm** (warm start), the function will restart the knot-placing strategy using the knots found in a previous call of the function. In this case, all parameters except **s** must be unchanged from that previous call. This warm start can save much time in searching for a satisfactory value of the smoothing factor $S$.
Constraint: **start** = **Nag_Cold** or **Nag_Warm**.

**m**

Input: $m$, the number of data points.
Constraint: **m** $\geq 4$.

**x[m]**

Input: **x**$[r-1]$ holds the value $x_r$ of the independent variable (abscissa) $x$, for $r = 1, 2, \ldots, m$.
Constraint: $x_1 < x_2 < \ldots < x_m$

**y[m]**

Input: **y**$[r-1]$ holds the value $y_r$ of the dependent variable (ordinate) $y$, for $r = 1, 2, \ldots, m$.

**weights[m]**

Input: **weights**$[r-1]$ holds the value $w_r$ of the weights, for $r = 1, 2, \ldots, m$. For advice on the choice of weights, see the Chapter Introduction.
Constraint: **weights**$[r-1] > 0$, for $r = 1, 2, \ldots, m$.

**s**

Input: the smoothing factor, $S$.
If $S = 0.0$, the function returns an interpolating spline.
If $S$ is smaller than ***machine precision***, it is assumed equal to zero.
For advice on the choice of $S$, see Sections 3 and 6.3.
Constraint: **s** $\geq 0.0$.

**nest**

Input: an over-estimate for the number, $n$, of knots required.
Constraint: **nest** $\geq 8$. In most practical situations, **nest** = **m**/2 is sufficient. **nest** never needs to be larger than **m**+4, the number of knots needed for interpolation (**s** = 0.0).

**fp**

Output: the sum of the squared weighted residuals, $\theta$, of the computed spline approximation. If **fp** = 0.0, this is an interpolating spline. **fp** should equal $S$ within a relative tolerance of 0.001 unless $n = 8$ when the spline has no interior knots and so is simply a cubic polynomial. For knots to be inserted, $S$ must be set to a value below the value of **fp** produced in this case.

**warmstartinf**

Input: Pointer to structure of type Nag_Comm with the following members:

**nag_w** - double *

Input: If the warm start option is used, the values **nag_w**[0],. . .,**nag_w**[**spline.n**−1] must be left unchanged from the previous call.

**nag_iw** - Integer *

Input: If the warm start option is used, the values **nag_iw**[0],. . .,**nag_iw**[**spline.n**−1] must be left unchanged from the previous call.

**spline**

Input/Output: Pointer to structure of type Nag_Spline with the following members:

**n** - Integer

Input: if the warm start option is used, the value of **spline.n** must be left unchanged from the previous call.
Output: the total number, $n$, of knots of the computed spline.

**lamda** - double *

Input: a pointer to which, if **start** = Nag_Cold, memory of size **nest** is internally allocated. If the warm start option is used, the values **spline.lamda**[0], **spline.lamda**[1], ..., **spline.lamda**[**spline.n**−1] must be left unchanged from the previous call.

Output: the knots of the spline i.e., the positions of the interior knots **spline.lamda**[4], **spline.lamda**[5],...,**spline.lamda**[**spline.n**−5] as well as the positions of the additional knots **spline.lamda**[0] = **spline.lamda** [1] = **spline.lamda**[2] = **spline.lamda**[3] = **x**[0] and **spline.lamda**[**spline.n**−4] = **spline.lamda**[**spline.n**−3] = **spline.lamda**[**spline.n**−2] = **spline.lamda**[**spline.n**−1] = **x**[**m**−1] needed for the B-spline representation.

**c** - double *

Output: a pointer to which, if **start** = Nag_Cold, memory of size **nest**−4 is internally allocated. **spline.c**[$i-1$] holds the coefficient $c_i$ of the B-spline $N_i(x)$ in the spline approximation $s(x)$, for $i = 1, 2, \ldots, n-4$.

**fail**

The NAG error parameter, see the Essential Introduction to the NAG C Library.

## 5. Error Indications and Warnings

**NE_BAD_PARAM**

On entry, parameter **start** had an illegal value.

**NE_INT_ARG_LT**

On entry, **m** must not be less than 4: **m** = ⟨*value*⟩.
On entry, **nest** must not be less than 8: **nest** = ⟨*value*⟩.

**NE_REAL_ARG_LT**

On entry, **s** must not be less than 0.0: **s** = ⟨*value*⟩.

**NE_WEIGHTS_NOT_POSITIVE**

On entry, the weights are not strictly positive: **weights**[⟨*value*⟩] = ⟨*value*⟩.

**NE_NOT_STRICTLY_INCREASING**

The sequence **x** is not strictly increasing: **x**[⟨*value*⟩] = ⟨*value*⟩ **x**[⟨*value*⟩] = ⟨*value*⟩.

**NE_SF_D_K_CONS**

On entry, **nest** = ⟨*value*⟩, **s** = ⟨*value*⟩, **m** = ⟨*value*⟩.
Constraint: **nest** ≥ **m** + 4 when **s** = 0.0.

**NE_ALLOC_FAIL**

Memory allocation failed.

**NE_ENUMTYPE_WARM**

**Start** has been set to Nag_Warm at the first call of this function. It must be set to Nag_Cold at the first call.

**NE_NUM_KNOTS_1D_GT**

The number of knots needed is greater than **nest**, **nest** = ⟨*value*⟩. If **nest** is already large, say **nest** > **m**/2, this may indicate that possibly **s** is too small: **s** = ⟨*value*⟩.

**NE_SPLINE_COEFF_CONV**

The iterative process has failed to converge. Possibly **s** is too small: **s** = ⟨*value*⟩.

If the function fails with an error exit of **NE_SPLINE_COEFF_CONV** or **NE_NUM_KNOTS_ID_GT**, a spline approximation is returned, but it fails to satisfy the fitting criterion (see (2) and (3) in Section 3) - perhaps by only a small amount, however.

## 6. Further Comments

### 6.1. Accuracy

On successful exit, the approximation returned is such that its weighted sum of squared residuals **fp** is equal to the smoothing factor $S$, up to a specified relative tolerance of 0.001 – except that if $n = 8$, **fp** may be significantly less than $S$: in this case the computed spline is simply a weighted least-squares polynomial approximation of degree 3, i.e., a spline with no interior knots.

### 6.2 Timing

The time taken for a call of nag_1d_spline_fit depends on the complexity of the shape of the data, the value of the smoothing factor $S$, and the number of data points. If nag_1d_spline_fit is to be called for different values of $S$, much time can be saved by setting **start** = **Nag_Warm** after the first call.

### 6.3. Choice of $S$

If the weights have been correctly chosen (see the Chapter Introduction), the standard deviation of $w_r y_r$ would be the same for all $r$, equal to $\sigma$, say. In this case, choosing the smoothing factor $S$ in the range $\sigma^2(m \pm \sqrt{2m})$, as suggested by Reinsch (1967), is likely to give a good start in the search for a satisfactory value. Otherwise, experimenting with different values of $S$ will be required from the start, taking account of the remarks in Section 3 .

In that case, in view of computation time and memory requirements, it is recommended to start with a very large value for $S$ and so determine the least-squares cubic polynomial; the value returned for **fp**, call it $\mathbf{fp}_0$, gives an upper bound for $S$. Then progressively decrease the value of $S$ to obtain closer fits – say by a factor of 10 in the beginning, i.e., $S = \mathbf{fp}_0/10$, $S = \mathbf{fp}_0/100$, and so on, and more carefully as the approximation shows more details.

The number of knots of the spline returned, and their location, generally depend on the value of $S$ and on the behaviour of the function underlying the data. However, if nag_1d_spline_fit is called with **start** = **Nag_Warm**, the knots returned may also depend on the smoothing factors of the previous calls. Therefore if, after a number of trials with different values of $S$ and **start** = **Nag_Warm**, a fit can finally be accepted as satisfactory, it may be worthwhile to call nag_1d_spline_fit once more with the selected value for $S$ but now using **start** = **Nag_Cold**. Often, nag_1d_spline_fit then returns an approximation with the same quality of fit but with fewer knots, which is therefore better if data reduction is also important.

### 6.4. Outline of Method Used

If $S = 0$, the requisite number of knots is known in advance, i.e. $n = m + 4$; the interior knots are located immediately as $\lambda_i = x_{i-2}$, for $i = 5, 6, \ldots, n-4$. The corresponding least-squares spline (see nag_1d_spline_fit_knots (e02bac)) is then an interpolating spline and therefore a solution of the problem.

If $S > 0$, a suitable knot set is built up in stages (starting with no interior knots in the case of a cold start but with the knot set found in a previous call if a warm start is chosen). At each stage, a spline is fitted to the data by least-squares (see nag_1d_spline_fit_knots (e02bac)) and $\theta$, the weighted sum of squares of residuals, is computed. If $\theta > S$, new knots are added to the knot set to reduce $\theta$ at the next stage. The new knots are located in intervals where the fit is particularly poor, their number depending on the value of $S$ and on the progress made so far in reducing $\theta$. Sooner or later, we find that $\theta \leq S$ and at that point the knot set is accepted. The function then goes on to compute the (unique) spline which has this knot set and which satisfies the full fitting criterion specified by (2) and (3). The theoretical solution has $\theta = S$. The function computes the spline by an iterative scheme which is ended when $\theta = S$ within a relative tolerance of 0.001. The main part of each iteration consists of a linear least-squares computation of special form, done in a similarly stable and efficient manner as in nag_1d_spline_fit_knots (e02bac).

An exception occurs when the function finds at the start that, even with no interior knots ($n = 8$), the least-squares spline already has its weighted sum of squares of residuals $\leq S$. In this case, since this spline (which is simply a cubic polynomial) also has an optimal value for the smoothness measure $\eta$, namely zero, it is returned at once as the (trivial) solution. It will usually mean that $S$ has been chosen too large.

For further details of the algorithm and its use, see Dierckx (1981).

### 6.5. Evaluation of Computed Spline

The value of the computed spline at a given value **x** may be obtained in the variable **sval** by the call:

```
e02bbc(x, &sval, &spline, &fail)
```

where **spline** is a structure of type Nag_Spline which is the output parameter of nag_1d_spline_fit.

The values of the spline and its first three derivatives at a given value **x** may be obtained in the array **sdif** of dimension at least 4 by the call:

```
e02bcc(derivs, x, sdif, &spline, &fail)
```

where, if **derivs** = **Nag_LeftDerivs**, left-hand derivatives are computed and, if **derivs** = **Nag_RightDerivs**, right-hand derivatives are calculated. The value of **derivs** is only relevant if **x** is an interior knot.

The value of the definite integral of the spline over the interval **x**[0] to **x**[**m**−1] can be obtained in the variable **sint** by the call:

```
e02bdc(&spline, &sint, &fail)
```

### 6.6. References

Dierckx P (1975) An Algorithm for Smoothing, Differentiating and Integration of Experimental Data Using Spline Functions *J. Comput. Appl. Math.* **1** 165–184.

Dierckx P (1981) *An Improved Algorithm for Curve Fitting with Spline Functions* Dept. Computer Science, K.U.Leuven, Report TW54.

Dierckx P (1982) A Fast Algorithm for Smoothing Data on a Rectangular Grid while using Spline Functions *SIAM J. Numer. Anal.* **19** 1286–1304.

Reinsch C H (1967) Smoothing by Spline Functions *Num. Math.* **10** 177–183.

## 7. See Also

nag_1d_spline_interpolant (e01bac)
nag_1d_spline_fit_knots (e02bac)
nag_1d_spline_evaluate (e02bbc)
nag_1d_spline_deriv (e02bcc)
nag_1d_spline_intg (e02bdc)

## 8. Example

This example program reads in a set of data values, followed by a set of values of $S$. For each value of $S$ it calls nag_1d_spline_fit to compute a spline approximation, and prints the values of the knots and the B-spline coefficients $c_i$.

The program includes code to evaluate the computed splines, by calls to nag_1d_spline_evaluate (e02bbc), at the points $x_r$ and at points mid-way between them. These values are not printed out, however; instead the results are illustrated by plots of the computed splines, together with the data points (indicated by $\times$) and the positions of the knots (indicated by vertical lines): the effect of decreasing $S$ can be clearly seen.

### 8.1. Program Text

```
/* nag_1d_spline_fit(e02bec) Example Program
 *
 * Copyright 1991 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 */

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

#define MMAX 50
#define NEST MMAX + 4
```

```
main()
{
  Integer m, r, j;
  double weights[MMAX], x[MMAX], y[MMAX];
  double s, fp, sp[2*MMAX-1], txr;
  Nag_Start start;
  Nag_Comm warmstartinf;
  Nag_Spline spline;

  Vprintf("e02bec Example Program Results\n");
  Vscanf("%*[^\n]"); /* Skip heading in data file */
  /* Input the number of data points, followed by the data
   * points x, the function values y and the weights w.
   */
  Vscanf("%ld",&m);
  if (m>0 && m<=MMAX)
    {
      start = Nag_Cold;
      for (r=0; r<m; r++)
        Vscanf("%lf%lf%lf",&x[r], &y[r], &weights[r]);
      /* Read in successive values of s until end of data file. */
      while(scanf("%lf",&s) !=EOF)
        {
          /* Determine the spline approximation. */
          e02bec(start, m, x, y, weights, s, (Integer)(NEST), &fp,
                 &warmstartinf, &spline, NAGERR_DEFAULT);
          /* Evaluate the spline at each x point and midway
           * between x points, saving the results in sp.
           */
          for (r=0; r<m; r++)
            e02bbc(x[r], &sp[r*2], &spline, NAGERR_DEFAULT);
          for (r=0; r<m-1; r++)
            {
              txr = (x[r] + x[r+1]) / 2;
              e02bbc(txr, &sp[r*2], &spline, NAGERR_DEFAULT);
            }
          /* Output the results. */
          Vprintf("\nCalling with smoothing factor s = %12.3e\n",s);
          Vprintf("\nNumber of distinct knots = %ld\n\n", spline.n-6);
          Vprintf("Distinct knots located at \n\n");
          for (j=3; j<spline.n-3; j++)
            Vprintf("%8.4f%s",spline.lamda[j],
                    (j-3)%6==5 || j==spline.n-4 ? "\n" : " ");
          Vprintf("\n\n    J      B-spline coeff c\n\n");
          for (j=0;  j<spline.n-4; ++j)
            Vprintf("  %3ld  %13.4f\n",j+1,spline.c[j]);
          Vprintf("\nWeighted sum of squared residuals fp = %12.3e\n",fp);
          if (fp == 0.0)
            Vprintf("The spline is an interpolating spline\n");
          else if (spline.n == 8)
            Vprintf("The spline is the weighted least-squares cubic\
 polynomial\n");
          start = Nag_Warm;
        }
      exit(EXIT_SUCCESS);
    }
  else
    {
      Vfprintf(stderr, "m is out of range: m = %5ld\n", m);
      exit(EXIT_FAILURE);
    }
}
```

### 8.2. Program Data

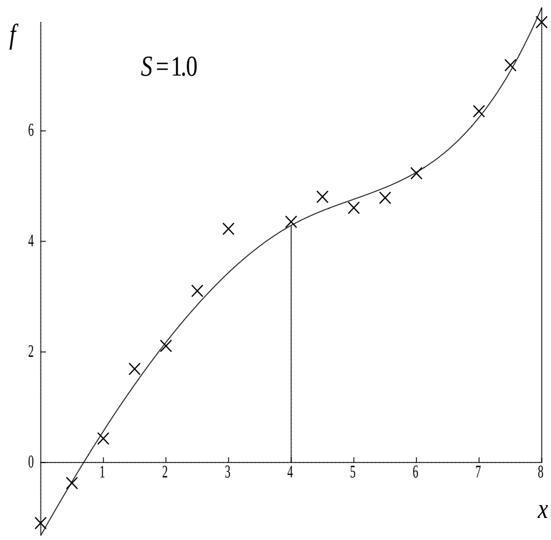
```
e02bec Example Program Data
  15
  0.0000E+00  -1.1000E+00   1.00
  5.0000E-01  -3.7200E-01   2.00
  1.0000E+00   4.3100E-01   1.50
  1.5000E+00   1.6900E+00   1.00
```

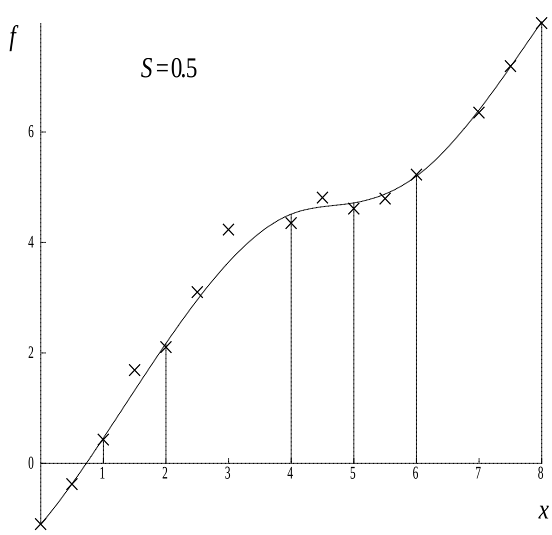
```
2.0000E+00   2.1100E+00   3.00
2.5000E+00   3.1000E+00   1.00
3.0000E+00   4.2300E+00   0.50
4.0000E+00   4.3500E+00   1.00
4.5000E+00   4.8100E+00   2.00
5.0000E+00   4.6100E+00   2.50
5.5000E+00   4.7900E+00   1.00
6.0000E+00   5.2300E+00   3.00
7.0000E+00   6.3500E+00   1.00
7.5000E+00   7.1900E+00   2.00
8.0000E+00   7.9700E+00   1.00
1.0
0.5
0.1
```

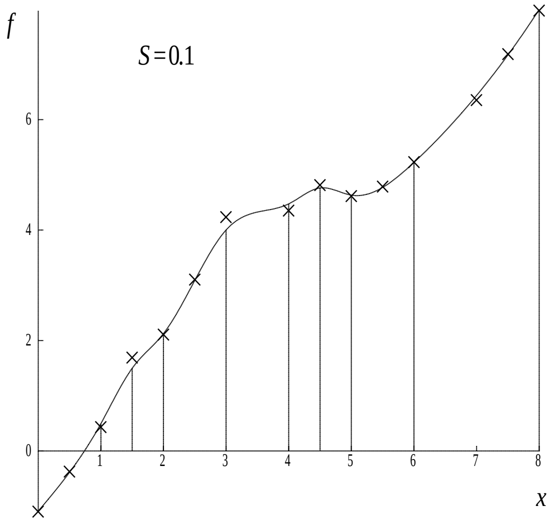
### 8.3. Program Results

```
e02bec Example Program Results

Calling with smoothing factor s =    1.000e+00

Number of distinct knots = 3

Distinct knots located at

  0.0000   4.0000   8.0000


    J      B-spline coeff c

    1        -1.3201
    2         1.3542
    3         5.5510
    4         4.7031
    5         8.2277

Weighted sum of squared residuals fp =    1.000e+00

Calling with smoothing factor s =    5.000e-01

Number of distinct knots = 7

Distinct knots located at

  0.0000   1.0000   2.0000   4.0000   5.0000   6.0000
  8.0000


    J      B-spline coeff c

    1        -1.1072
    2        -0.6571
    3         0.4350
    4         2.8061
    5         4.6824
    6         4.6416
    7         5.1976
    8         6.9008
    9         7.9979

Weighted sum of squared residuals fp =    5.001e-01

Calling with smoothing factor s =    1.000e-01

Number of distinct knots = 10

Distinct knots located at

  0.0000   1.0000   1.5000   2.0000   3.0000   4.0000
  4.5000   5.0000   6.0000   8.0000
```

```
   J      B-spline coeff c

   1        -1.0900
   2        -0.6422
   3         0.0369
   4         1.6353
   5         2.1274
   6         4.5526
   7         4.2225
   8         4.9108
   9         4.4159
  10         5.4794
  11         6.8308
  12         7.9935

Weighted sum of squared residuals fp =    1.000e-01
```




f
S=0.1
6
4
2
0
1
2
3
4
5
6
7
8
x