# NAG Library Function Document

# nag_smooth_spline_estim (g10acc)

## 1 Purpose

nag_smooth_spline_estim (g10acc) estimates the values of the smoothing argument and fits a cubic smoothing spline to a set of data.

## 2 Specification

```
#include <nag.h>
#include <nagg10.h>
```

```
void nag_smooth_spline_estim (Nag_SmoothParamMethods method, Integer n,
     const double x[], const double y[], const double weights[],
     double yhat[], double coeff[], double *rss, double *df, double res[],
     double h[], double *crit, double *rho, double u, double tol,
     Integer maxcal, NagError *fail)
```

## 3 Description

For a set of $n$ observations $(x_i, y_i)$, for $i = 1, 2, \ldots, n$, the spline provides a flexible smooth function for situations in which a simple polynomial or nonlinear regression model is not suitable.

Cubic smoothing splines arise as the unique real-valued solution function, $f$, with absolutely continuous first derivative and squared-integrable second derivative, which minimizes:

$$\sum_{i=1}^{n} w_i \{y_i - f(x_i)\}^2 + \rho \int_{-\infty}^{\infty} \{f''(x)\}^2 dx,$$

where $w_i$ is the (optional) weight for the $i$th observation and $\rho$ is the smoothing argument. This criterion consists of two parts: the first measures the fit of the curve and the second the smoothness of the curve. The value of the smoothing argument $\rho$ weights these two aspects; larger values of $\rho$ give a smoother fitted curve but, in general, a poorer fit. For details of how the cubic spline can be fitted see Hutchinson and de Hoog (1985) and Reinsch (1967).

The fitted values, $\hat{y} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n)^{\mathrm{T}}$, and weighted residuals, $r_i$, can be written as:

$$\hat{y} = Hy \quad \text{and} \quad r_i = \sqrt{w_i}(y_i - \hat{y}_i)$$

for a matrix $H$. The residual degrees of freedom for the spline is trace$(I - H)$ and the diagonal elements of $H$ are the leverages.

The argument $\rho$ can be estimated in a number of ways.

1. The degrees of freedom for the spline can be specified, i.e., find $\rho$ such that trace$(H) = \nu_0$ for given $\nu_0$.
2. Minimize the cross-validation (CV), i.e., find $\rho$ such that the CV is minimized, where

$$\mathrm{CV} = \frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} \left[\frac{r_i}{1 - h_{ii}}\right]^2.$$

3. Minimize the generalized cross-validation (GCV), i.e., find $\rho$ such that the GCV is minimized, where

$$\mathrm{GCV} = \frac{n^2}{\sum_{i=1}^{n} w_i} \left[ \frac{\sum_{i=1}^{n} r_i^2}{\left( \sum_{i=1}^{n} (1 - h_{ii}) \right)^2} \right].$$

nag_smooth_spline_estim (g10acc) requires the $x_i$ to be strictly increasing. If two or more observations have the same $x_i$ value then they should be replaced by a single observation with $y_i$ equal to the (weighted) mean of the $y$ values and weight, $w_i$, equal to the sum of the weights. This operation can be performed by nag_order_data (g10zac).

The algorithm is based on Hutchinson (1986).

## 4 References

Hastie T J and Tibshirani R J (1990) *Generalized Additive Models* Chapman and Hall

Hutchinson M F (1986) Algorithm 642: A fast procedure for calculating minimum cross-validation cubic smoothing splines *ACM Trans. Math. Software* **12** 150–153

Hutchinson M F and de Hoog F R (1985) Smoothing noisy data with spline functions *Numer. Math.* **47** 99–106

Reinsch C H (1967) Smoothing by spline functions *Numer. Math.* **10** 177–183

## 5 Arguments

1: **method** – Nag_SmoothParamMethods *Input*

*On entry*: indicates whether the smoothing argument is to be found by minimization of the CV or GCV functions, or by finding the smoothing argument corresponding to a specified degrees of freedom value.

**method** = Nag_SmoothParamCV
Cross-validation is used.

**method** = Nag_SmoothParamDF
The degrees of freedom are specified.

**method** = Nag_SmoothParamGCV
Generalized cross-validation is used.

*Constraint*: **method** = Nag_SmoothParamCV, Nag_SmoothParamDF or Nag_SmoothParamGCV.

2: **n** – Integer *Input*

*On entry*: the number of observations, $n$.

*Constraint*: **n** $\geq 3$.

3: **x**[**n**] – const double *Input*

*On entry*: the distinct and ordered values $x_i$, for $i = 1, 2, \ldots, n$.

*Constraint*: **x**$[i-1] <$ **x**$[i]$, for $i = 1, 2, \ldots, n-1$.

4: **y**[**n**] – const double *Input*

*On entry*: the values $y_i$, for $i = 1, 2, \ldots, n$.

5: **weights**[**n**] – const double *Input*

*On entry*: **weights** must contain the $n$ weights, if they are required. Otherwise, **weights** must be set to **NULL**.

*Constraint*: if **weights** are required, then **weights**$[i-1] > 0.0$, for $i = 1, 2, \ldots, n$.

6: **yhat**[**n**] – double *Output*

*On exit*: the fitted values, $\hat{y}_i$, for $i = 1, 2, \ldots, n$.

7: **coeff**[(**n** − **1**) × **3**] – double *Output*

*On exit*: the spline coefficients. More precisely, the value of the spline approximation at $t$ is given by $((\mathbf{coeff}[(i-1) \times (n-1) + 2] \times d + \mathbf{coeff}[(i-1) \times (n-1) + 1]) \times d$ $+ \mathbf{coeff}[(i-1) \times (n-1)]) \times d + \hat{y}_i$, where $x_i \leq t < x_{i+1}$ and $d = t - x_i$.

8: **rss** – double * *Output*

*On exit*: the (weighted) residual sum of squares.

9: **df** – double * *Output*

*On exit*: the residual degrees of freedom. If **method** = Nag_SmoothParamDF, this will be $n -$ **crit** to the required accuracy.

10: **res**[**n**] – double *Output*

*On exit*: the (weighted) residuals, $r_i$, for $i = 1, 2, \ldots, n$.

11: **h**[**n**] – double *Output*

*On exit*: the leverages, $h_{ii}$, for $i = 1, 2, \ldots, n$.

12: **crit** – double * *Input/Output*

*On entry*: if **method** = Nag_SmoothParamDF, the required degrees of freedom for the spline.

If **method** = Nag_SmoothParamCV or Nag_SmoothParamGCV, **crit** need not be set.

*Constraint*: $2.0 <$ **crit** $\leq$ **n**.

*On exit*: if **method** = Nag_SmoothParamCV, the value of the cross-validation, or if **method** = Nag_SmoothParamGCV, the value of the generalized cross-validation function, evaluated at the value of $\rho$ returned in **rho**.

13: **rho** – double * *Output*

*On exit*: the smoothing argument, $\rho$.

14: **u** – double *Input*

*On entry*: the upper bound on the smoothing argument. See Section 9 for details on how this argument is used.

*Suggested value*: **u** = 1000.0.

*Constraint*: **u** > **tol**.

15: **tol** – double *Input*

*On entry*: the accuracy to which the smoothing argument **rho** is required. **tol** should be preferably not much less than $\sqrt{\epsilon}$, where $\epsilon$ is the ***machine precision***.

*Constraint*: **tol** ≥ ***machine precision***.

16: **maxcal** – Integer *Input*

*On entry*: the maximum number of spline evaluations to be used in finding the value of $\rho$.

*Suggested value*: **maxcal** $= 30$.

*Constraint*: **maxcal** $\geq 3$.

17: **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_2_REAL_ARG_LE**

On entry, **u** $= \langle value \rangle$ while **tol** $= \langle value \rangle$. These arguments must satisfy **u** $>$ **tol**.

**NE_ALLOC_FAIL**

Dynamic memory allocation failed.

**NE_BAD_PARAM**

On entry, argument **method** had an illegal value.

**NE_G10AC_ACC**

A solution to the accuracy given by **tol** has not been achieved in **maxcal** iterations. Try increasing the value of **tol** and/or **maxcal**.

**NE_G10AC_CG_RHO**

**method** = Nag_SmoothParamCV or Nag_SmoothParamGCV and the optimal value of **rho** $>$ **u**. Try a larger value of **u**.

**NE_G10AC_DF_RHO**

**method** = Nag_SmoothParamDF and the required value of **rho** for specified degrees of freedom $>$ **u**. Try a larger value of **u**.

**NE_G10AC_DF_TOL**

**method** = Nag_SmoothParamDF and the accuracy given by **tol** cannot be achieved. Try increasing the value of **tol**.

**NE_INT_ARG_LT**

On entry, **maxcal** $= \langle value \rangle$.
Constraint: **maxcal** $\geq 3$.

On entry, **n** $= \langle value \rangle$.
Constraint: **n** $\geq 3$.

**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_STRICTLY_INCREASING**

The sequence **x** is not strictly increasing: **x**$[\langle value \rangle] = \langle value \rangle$, **x**$[\langle value \rangle] = \langle value \rangle$.

**NE_REAL**

On entry, **crit** $= \langle value \rangle$.
Constraint: **crit** $> 2.0$, if **method** $=$ Nag_SmoothParamDF.

**NE_REAL_ARRAY_CONS**

On entry, **weights**$[\langle value \rangle] = \langle value \rangle$.
Constraint: **weights**$[i] > 0$, for $i = 0, 1, \ldots, n - 1$.

**NE_REAL_INT_ARG_CONS**

On entry, **crit** $= \langle value \rangle$ and **n** $= \langle value \rangle$. These arguments must satisfy **crit** $\leq$ **n**, if **method** $=$ Nag_SmoothParamDF.

**NE_REAL_MACH_PREC**

On entry, **tol** $= \langle value \rangle$, ***machine precision***(nag_machine_precision) $= \langle value \rangle$.
Constraint: **tol** $\geq$ ***machine precision***.

## 7 Accuracy

When minimizing the cross-validation or generalized cross-validation, the error in the estimate of $\rho$ should be within $\pm 3 \times$ (**tol** $\times$ **rho** $+$ **tol**). When finding $\rho$ for a fixed number of degrees of freedom the error in the estimate of $\rho$ should be within $\pm 2 \times$ **tol** $\times \max(1,$ **rho**$)$.

Given the value of $\rho$, the accuracy of the fitted spline depends on the value of $\rho$ and the position of the $x$ values. The values of $x_i - x_{i-1}$ and $w_i$ are scaled and $\rho$ is transformed to avoid underflow and overflow problems.

## 8 Parallelism and Performance

nag_smooth_spline_estim (g10acc) is not threaded in any implementation.

## 9 Further Comments

The time to fit the spline for a given value of $\rho$ is of order $n$.

When finding the value of $\rho$ that gives the required degrees of freedom, the algorithm examines the interval 0.0 to **u**. For small degrees of freedom the value of $\rho$ can be large, as in the theoretical case of two degrees of freedom when the spline reduces to a straight line and $\rho$ is infinite. If the CV or GCV is to be minimized then the algorithm searches for the minimum value in the interval 0.0 to **u**. If the function is decreasing in that range then the boundary value of **u** will be returned. In either case, the larger the value of **u** the more likely is the interval to contain the required solution, but the process will be less efficient.

Regression splines with a small ($< n$) number of knots can be fitted by nag_1d_spline_fit_knots (e02bac) and nag_1d_spline_fit (e02bec).

## 10 Example

The data, given by Hastie and Tibshirani (1990), is the age, $x_i$, and C-peptide concentration (pmol/ml), $y_i$, from a study of the factors affecting insulin-dependent diabetes mellitus in children. The data is input, reduced to a strictly ordered set by nag_order_data (g10zac) and a spline with 5 degrees of freedom is fitted by nag_smooth_spline_estim (g10acc). The fitted values and residuals are printed.

### 10.1 Program Text

```
/* nag_smooth_spline_estim (g10acc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg10.h>

int main(void)
{
  Integer exit_status = 0, i, maxcal, n, nord;
  Nag_SmoothParamMethods method;
  Nag_Boolean weight;
  char nag_enum_arg[40];
  double crit, df, rho, rss, tol, u;
  double *coeff = 0, *h = 0, *res = 0, *weights = 0, *wtptr;
  double *wwt = 0, *x = 0, *xord = 0, *y = 0, *yhat = 0;
  double *yord = 0;
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_smooth_spline_estim (g10acc) Example Program Results\n");

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

#ifdef _WIN32
  scanf_s("%" NAG_IFMT "", &n);
#else
  scanf("%" NAG_IFMT "", &n);
#endif
  if (!(x = NAG_ALLOC(n, double))
      || !(y = NAG_ALLOC(n, double))
      || !(weights = NAG_ALLOC(n, double))
      || !(yhat = NAG_ALLOC(n, double))
      || !(coeff = NAG_ALLOC((n - 1) * 3, double))
      || !(res = NAG_ALLOC(n, double))
      || !(h = NAG_ALLOC(n, double))
      || !(wwt = NAG_ALLOC(n, double))
      || !(yord = NAG_ALLOC(n, double))
      || !(xord = NAG_ALLOC(n, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

#ifdef _WIN32
  scanf_s("%39s", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%39s", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  method = (Nag_SmoothParamMethods) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%39s", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
```

```
#else
  scanf("%39s", nag_enum_arg);
#endif
  weight = (Nag_Boolean) nag_enum_name_to_value(nag_enum_arg);

  if (!weight) {
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
      scanf_s("%lf %lf", &x[i - 1], &y[i - 1]);
#else
      scanf("%lf %lf", &x[i - 1], &y[i - 1]);
#endif
    wtptr = 0;
  }
  else {
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
      scanf_s("%lf %lf %lf", &x[i - 1], &y[i - 1], &weights[i - 1]);
#else
      scanf("%lf %lf %lf", &x[i - 1], &y[i - 1], &weights[i - 1]);
#endif
    wtptr = weights;
  }

#ifdef _WIN32
  scanf_s("%lf %lf %" NAG_IFMT " %lf", &u, &tol, &maxcal, &crit);
#else
  scanf("%lf %lf %" NAG_IFMT " %lf", &u, &tol, &maxcal, &crit);
#endif
  /* Sort data, removing ties and weighting accordingly */
  /* nag_order_data (g10zac).
   * Reorder data to give ordered distinct observations
   */
  nag_order_data(n, x, y, wtptr, &nord, xord, yord, wwt, &rss, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_order_data (g10zac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
  /* Fit cubic spline */
  /* nag_smooth_spline_estim (g10acc).
   * Fit cubic smoothing spline, smoothing parameter estimated
   */
  nag_smooth_spline_estim(method, nord, xord, yord, wwt, yhat, coeff, &rss,
                          &df, res, h, &crit, &rho, u, tol, maxcal, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_smooth_spline_estim (g10acc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Print results */
  printf("\n");
  printf("%s%10.2f\n", " Residual sum of squares = ", rss);
  printf("%s%10.2f\n", " Degrees of freedom = ", df);
  printf("%s%10.2f\n", " rho = ", rho);
  printf("\n");
  printf("\n%s%s%s\n%s%s%s\n", "     Input data",
         "                ",
         "Output results",
         "   i    x        y   ", "          ", "yhat     h");

  for (i = 1; i <= nord; ++i)
    printf("%4" NAG_IFMT " %8.3f %8.3f      %8.3f %8.3f\n",
           i, xord[i - 1], yord[i - 1], yhat[i - 1], h[i - 1]);

END:
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(weights);
```

```
  NAG_FREE(yhat);
  NAG_FREE(coeff);
  NAG_FREE(res);
  NAG_FREE(h);
  NAG_FREE(wwt);
  NAG_FREE(yord);
  NAG_FREE(xord);

  return exit_status;
}
```

## 10.2 Program Data

```
nag_smooth_spline_estim (g10acc) Example Program Data
43
Nag_SmoothParamDF Nag_FALSE
 5.2 4.8    8.8 4.1  10.5 5.2  10.6 5.5  10.4 5.0
 1.8 3.4   12.7 3.4  15.6 4.9   5.8 5.6   1.9 3.7
 2.2 3.9    4.8 4.5   7.9 4.8   5.2 4.9   0.9 3.0
11.8 4.6    7.9 4.8  11.5 5.5  10.6 4.5   8.5 5.3
11.1 4.7   12.8 6.6  11.3 5.1   1.0 3.9  14.5 5.7
11.9 5.1    8.1 5.2  13.8 3.7  15.5 4.9   9.8 4.8
11.0 4.4   12.4 5.2  11.1 5.1   5.1 4.6   4.8 3.9
 4.2 5.1    6.9 5.1  13.2 6.0   9.9 4.9  12.5 4.1
13.2 4.6    8.9 4.9  10.8 5.1
10000  0.001  40  12.0
```

## 10.3 Program Results

```
nag_smooth_spline_estim (g10acc) Example Program Results

 Residual sum of squares =      10.35
 Degrees of freedom =      25.00
 rho =       2.68


     Input data                Output results
    i    x        y            yhat     h
    1    0.900    3.000        3.373    0.534
    2    1.000    3.900        3.406    0.427
    3    1.800    3.400        3.642    0.313
    4    1.900    3.700        3.686    0.313
    5    2.200    3.900        3.839    0.448
    6    4.200    5.100        4.614    0.564
    7    4.800    4.200        4.576    0.442
    8    5.100    4.600        4.715    0.189
    9    5.200    4.850        4.783    0.407
   10    5.800    5.600        5.193    0.455
   11    6.900    5.100        5.184    0.592
   12    7.900    4.800        4.958    0.530
   13    8.100    5.200        4.931    0.234
   14    8.500    5.300        4.845    0.245
   15    8.800    4.100        4.763    0.271
   16    8.900    4.900        4.748    0.292
   17    9.800    4.800        4.850    0.301
   18    9.900    4.900        4.875    0.276
   19   10.400    5.000        4.970    0.173
   20   10.500    5.200        4.977    0.154
   21   10.600    5.000        4.979    0.285
   22   10.800    5.100        4.970    0.136
   23   11.000    4.400        4.961    0.137
   24   11.100    4.900        4.964    0.284
   25   11.300    5.100        4.975    0.162
   26   11.500    5.500        4.975    0.186
   27   11.800    4.600        4.930    0.213
   28   11.900    5.100        4.911    0.220
   29   12.400    5.200        4.852    0.206
   30   12.500    4.100        4.857    0.196
   31   12.700    3.400        4.900    0.189
   32   12.800    6.600        4.932    0.193
```

```
  33   13.200    5.300           4.955    0.488
  34   13.800    3.700           4.797    0.408
  35   14.500    5.700           5.076    0.559
  36   15.500    4.900           4.979    0.445
  37   15.600    4.900           4.946    0.535
```