NAG Library Function Document

nag_smooth_spline_fit (g10abc)

1 Purpose

nag_smooth_spline_fit (g10abc) fits a cubic smoothing spline for a given smoothing parameter.

2 Specification

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

void nag_smooth_spline_fit (Nag_SmoothFitType mode, Integer n,
                    const double x[], const double y[], const double wt[], double rho,
                    double yhat[], double c[], Integer pdc, double *rss, double *df,
                    double res[], double h[], double comm[], NagError *fail)
```

3 Description

nag_smooth_spline_fit (g10abc) fits a cubic smoothing spline to 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 unsuitable.

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 parameter. 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 parameter \( \rho \) 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 estimated see Hutchinson and de Hoog (1985) and Reinsch (1967).

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

\[
\hat{y} = H\gamma \quad \text{and} \quad r_i = \sqrt{w_i} (y_i - \hat{y}_i)
\]

for a matrix \( H \). The residual degrees of freedom for the spline is \( \text{trace}(I - H) \) and the diagonal elements of \( H \), \( h_{ii} \), are the leverages.

The parameter \( \rho \) can be chosen in a number of ways. The fit can be inspected for a number of different values of \( \rho \). Alternatively the degrees of freedom for the spline, which determines the value of \( \rho \), can be specified, or the (generalized) cross-validation can be minimized to give \( \rho \); see nag_smooth_spline_estim (g10acc) for further details.

nag_smooth_spline_fit (g10abc) 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 computation is split into three phases.

(i) Compute matrices needed to fit spline.

(ii) Fit spline for a given value of \( \rho \).

(iii) Compute spline coefficients.
When fitting the spline for several different values of \( \rho \), phase (i) need only be carried out once and then phase (ii) repeated for different values of \( \rho \). If the spline is being fitted as part of an iterative weighted least squares procedure phases (i) and (ii) have to be repeated for each set of weights. In either case, phase (iii) will often only have to be performed after the final fit has been computed.

The algorithm is based on Hutchinson (1986).

### 4 References


### 5 Arguments

1: \( \text{mode} \) – Nag_SmoothFitType

*Input*

*On entry:* indicates in which mode the function is to be used.

\( \text{mode} = \text{Nag_SmoothFitPartial} \)

Initialization and fitting is performed. This partial fit can be used in an iterative weighted least squares context where the weights are changing at each call to nag_smooth_spline_fit (g10abc) or when the coefficients are not required.

\( \text{mode} = \text{Nag_SmoothFitQuick} \)

Fitting only is performed. Initialization must have been performed previously by a call to nag_smooth_spline_fit (g10abc) with \( \text{mode} = \text{Nag_SmoothFitPartial} \). This quick fit may be called repeatedly with different values of \( \rho \) without re-initialization.

\( \text{mode} = \text{Nag_SmoothFitFull} \)

Initialization and full fitting is performed and the function coefficients are calculated.

*Constraint:* \( \text{mode} = \text{Nag_SmoothFitPartial} \), \( \text{Nag_SmoothFitQuick} \) or \( \text{Nag_SmoothFitFull} \).

2: \( n \) – Integer

*Input*

*On entry:* \( n \), the number of distinct observations.

*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: \( \text{wt}[n] \) – const double

*Input*

*On entry:* optionally, the \( n \) weights.

If weights are not provided then \( \text{wt} \) must be set to \( \text{NULL} \), in which case unit weights are assumed.

*Constraint:* \( \text{wt}[i - 1] > 0.0 \), for \( i = 1, 2, \ldots, n \).
rho – double 

On entry: \( \rho \), the smoothing parameter.

Constraint: \( \rho \geq 0.0 \).

yhat[n] – double 

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

c[pdc × 3] – double 

Note: the \((i, j)\)th element of the matrix \( C \) is stored in \( c[(j - 1) \times \text{pdc} + i - 1] \).

On entry: if \( \text{mode} = \text{Nag_SmoothFitQuick} \), \( c \) must be unaltered from the previous call to \text{nag_smooth_spline_fit (g10abc)} with \( \text{mode} = \text{Nag_SmoothFitPartial} \). Otherwise \( c \) need not be set.

On exit: if \( \text{mode} = \text{Nag_SmoothFitFull} \), \( c \) contains the spline coefficients. More precisely, the value of the spline at \( t \) is given by \((c[2 \times \text{pdc} + i - 1] \times d + c[1 \times \text{pdc} + i - 1]) \times d + c[i - 1] \times d + \hat{y}_i \), where \( x_i \leq t < x_{i+1} \) and \( d = t - x_i \).

If \( \text{mode} = \text{Nag_SmoothFitPartial} \) or \( \text{Nag_SmoothFitQuick} \), \( c \) contains information that will be used in a subsequent call to \text{nag_smooth_spline_fit (g10abc)} with \( \text{mode} = \text{Nag_SmoothFitQuick} \).

pdc – Integer 

On entry: the stride separating matrix row elements in the array \( c \).

Constraint: \( \text{pdc} \geq n - 1 \).

rss – double * 

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

df – double * 

On exit: the residual degrees of freedom.

res[n] – double 

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

h[n] – double 

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

comm[9 × n + 14] – double 

On entry: if \( \text{mode} = \text{Nag_SmoothFitQuick} \), \( \text{comm} \) must be unaltered from the previous call to \text{nag_smooth_spline_fit (g10abc)} with \( \text{mode} = \text{Nag_SmoothFitPartial} \). Otherwise \( \text{comm} \) need not be set.

On exit: if \( \text{mode} = \text{Nag_SmoothFitPartial} \) or \( \text{Nag_SmoothFitQuick} \), \( \text{comm} \) contains information that will be used in a subsequent call to \text{nag_smooth_spline_fit (g10abc)} with \( \text{mode} = \text{Nag_SmoothFitQuick} \).

fail – NagError * 

The NAG error argument (see Section 3.6 in the Essential Introduction).
6 Error Indicators and Warnings

NE_ALLOC_FAIL
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_ARRAY_SIZE
On entry, pdc = ⟨value⟩ and n = ⟨value⟩.
Constraint: pdc ≥ n − 1.

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

NE_INT
On entry, n = ⟨value⟩.
Constraint: n ≥ 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.
An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE
Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_NOT_STRICTLY_INCREASING
On entry, x is not a strictly ordered array.

NE_REAL
On entry, rho = ⟨value⟩.
Constraint: rho ≥ 0.0.

NE_WEIGHTS_NOT_POSITIVE
On entry, at least one element of wt ≤ 0.0.

7 Accuracy
Accuracy depends on the value of ρ and the position of the x values. The values of xi − xi−1 and wi are
scaled and ρ is transformed to avoid underflow and overflow problems.

8 Parallelism and Performance
Not applicable.

9 Further Comments
The time taken by nag_smooth_spline_fit (g10abc) is of order n.
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 series of splines fit using a range of values for the smoothing parameter, \(\rho\).

10.1 Program Text

```c
/* nag_smooth_spline_fit (g10abc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 6a revised, 2001. */

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

int main(void)
{
  Integer exit_status = 0, i, n, nord;
  double df, rho, rss;
  double *coeff = 0, *comm_ar = 0, *h = 0, *res = 0, *weights = 0;
  double *wtptar, *wwt = 0, *x = 0, *xord = 0, *y = 0, *yhat = 0;
  double *yord = 0;
  char nag_enum_arg[40];
  Nag_SmoothFitType mode;
  Nag_Boolean weight;
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_smooth_spline_fit (g10abc) Example Program Results\n");

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

  if (!(coeff = NAG_ALLOC((n-1)*3, double))
      || !(h = NAG_ALLOC(n, double))
      || !(res = NAG_ALLOC(n, double))
      || !(x = NAG_ALLOC(n, double))
      || !(y = NAG_ALLOC(n, double))
      || !(weights = NAG_ALLOC(n, double))
      || !(xord = NAG_ALLOC(n, double))
      || !(yord = NAG_ALLOC(n, double))
      || !(wwt = NAG_ALLOC(n, double))
      || !(yhat = NAG_ALLOC(n, double))
      || !(comm_ar = NAG_ALLOC(9*n+14, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  #ifdef _WIN32
    scanf_s("%39s", nag_enum_arg, _countof(nag_enum_arg));
  #else
    scanf("%39s", nag_enum_arg, _countof(nag_enum_arg));
  #endif

END:
exit_status = 0;
return exit_status;
}
```

---

Mark 25

**g10abc**

---
```c
scanf("%39s", nag_enum_arg);
#endif
/* nag_enum_name_to_value (x04nac).
* Converts NAG enum member name to value
*/
mode = (Nag_SmoothFitType) nag_enum_name_to_value(nag_enum_arg);
#endif _WIN32
scanf_s("%39s", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf("%39s", nag_enum_arg);
#endif
weight = (Nag_Boolean) nag_enum_name_to_value(nag_enum_arg);
#endif _WIN32
scanf_s("%lf", &rho);
#else
scanf("%lf", &rho);
#endif
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;
    }
/* Sort data into increasing X and */
/* remove tied observations and weight 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_fit (g10abc).
* Fit cubic smoothing spline, smoothing parameter given */
    nag_smooth_spline_fit(mode, nord, xord, yord, wwt, rho, yhat, coeff, 
                          &rss, &df, res, h, comm_ar, &fail);
if (fail.code != NE_NOERROR)
    {  
        printf("Error from nag_smooth_spline_fit (g10abc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
/* Print results */
printf("\n");
printf("%s%10.3f\n", " rho = ", rho);
printf("\n");
printf("%s%10.3f\n", " Residual sum of squares = ", rss);
```
printf("%s%10.3f\n", " Degrees of freedom = ", df);
printf("\n");
printf("%s\n", " Ordered input data Output results");
printf("\n");
printf("%s\n", " x y Fitted Values");
printf("\n");
for (i = 1; i <= nord; ++i)
{
    printf("%8.4f %8.4f %8.4f\n",
           xord[i - 1],
           yord[i - 1],
           yhat[i - 1]);
}

END:
NAG_FREE(coeff);
NAG_FREE(h);
NAG_FREE(res);
NAG_FREE(x);
NAG_FREE(y);
NAG_FREE(weights);
NAG_FREE(xord);
NAG_FREE(yord);
NAG_FREE(wwt);
NAG_FREE(yhat);
NAG_FREE(comm_ar);
return exit_status;
}

10.2 Program Data
nag_smooth_spline_fit (g10abc) Example Program Data
43
Nag_SmoothFitFull Nag_FALSE
10.0
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

10.3 Program Results
nag_smooth_spline_fit (g10abc) Example Program Results
rho = 10.000
Residual sum of squares = 11.288
Degrees of freedom = 27.785
Ordered input data Output results
   x   y   Fitted Values
0.9000 3.0000 3.3674
1.0000 3.9000 3.4008
1.8000 3.4000 3.6642
1.9000 3.7000 3.7016
2.2000 3.9000 3.8214
4.2000 5.1000 4.5265
4.8000 4.2000 4.6471
5.1000 4.6000 4.7561
5.2000 4.8500 4.7993
5.8000 5.6000 5.0458
6.9000 5.1000 5.1204
Example Program
Cubic Smoothing Spline
Study of the factors affecting insulin-dependent diabetes mellitus in children
Hastie and Tibshirani (1990)

raw data
ρ = 1
ρ = 10
ρ = 100