NAG C 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

#include <nag.h>
#include <nagl10.h>

void nag_smooth_spline_fit(Nag_SmoothFitType mode, Integer n,
                        const double x[], const double y[], const double weights[],
                        double rho, double yhat[], double coeff[], double *rss, double *df,
                        double res[], double h[], double comm_ar[], NagError *fail)

3 Description

nag_smooth_spline_fit 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 non-linear 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 minimises:

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

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:

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

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 (generalised) cross-validation can be minimised to give \( \rho \); see nag_smooth_spline_estim (g10acc) for further details.

nag_smooth_spline_fit 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.

1. Compute matrices needed to fit spline.
2. Fit spline for a given value of \( \rho \).
3. Compute spline coefficients.
When fitting the spline for several different values of \( \rho \), phase (1) need only be carried out once and then phase (2) repeated for different values of \( \rho \). If the spline is being fitted as part of an iterative weighted least-squares procedure phases (1) and (2) have to be repeated for each set of weights. In either case, phase (3) will often only have to be performed after the final fit has been computed.

The algorithm is based on Hutchinson (1986).

## 4 Parameters

1. **mode** – Nag_SmoothFitType  
   *Input*  
   
   On entry: indicates in which mode the routine is to be used.

   If **mode** = Nag_SmoothFitPartial, initialisation 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 or when the coefficients are not required.

   If **mode** = Nag_SmoothFitQuick, fitting only is performed. Initialisation must have been performed previously by a call to nag_smooth_spline_fit with **mode** = Nag_SmoothFitPartial. This Quick fit may be called repeatedly with different values of **rho** without re-initialisation.

   If **mode** = Nag_SmoothFitFull, initialisation and Full fitting is performed and the function coefficients are calculated.

   *Constraint:* **mode** = Nag_SmoothFitPartial, Nag_SmoothFitQuick or Nag_SmoothFitFull.

2. **n** – Integer  
   *Input*  
   
   On entry: the number of distinct 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 the null pointer (double*) \( 0 \).

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

6. **rho** – double  
   *Input*  
   
   On entry: the smoothing parameter, \( \rho \).

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

7. **yhat[n]** – double  
   *Output*  
   
   On exit: the fitted values, \( \hat{y}_i \), for \( i = 1, 2, \ldots, n \).

8. **coeff[(n-1)*3]** – double  
   *Input/Output*  
   
   On entry: if **mode** = Nag_SmoothFitQuick, **coeff** must be unaltered from the previous call to nag_smooth_spline_fit with **mode** = Nag_SmoothFitPartial. Otherwise **coeff** need not be set.

   On exit: if **mode** = Nag_SmoothFitFull, **coeff** contains the spline coefficients. More precisely, the value of the spline at \( t \) is given by \( ((\text{coeff}[i-1] \times (n-1)+2] \times d + \text{coeff}[i-1] \times (n-1)+1) \times d + \text{coeff}[i-1] \times (n-1)+1) \times d + \hat{y}_i \), where \( x_i \leq t < x_{i+1} \) and \( d = t - x_i \).
If `mode = Nag_SmoothFitPartial` or `Nag_SmoothFitQuick`, `coeff` contains information that will be used in a subsequent call to `nag_smooth_spline_fit` with `mode = Nag_SmoothFitQuick`.

9: `rss – double *` **Output**
   On exit: the (weighted) residual sum of squares.

10: `df – double *` **Output**
   On exit: the residual degrees of freedom.

11: `res[n] – double` **Output**
   On exit: the (weighted) residuals, $r_i$, for $i = 1, 2, \ldots, n$.

12: `h[n] – double` **Output**
   On exit: the leverages, $h_{ii}$, for $i = 1, 2, \ldots, n$.

   On entry: if `mode = Nag_SmoothFitQuick`, `comm_ar` must be unaltered from the previous call to `nag_smooth_spline_fit` with `mode = Nag_SmoothFitPartial`. Otherwise `comm_ar` is used as workspace and need not be set.
   On exit: if `mode = Nag_SmoothFitPartial` or `Nag_SmoothFitQuick`, `comm_ar` contains information that will be used in a subsequent call to `nag_smooth_spline_fit` with `mode = Nag_SmoothFitQuick`.

14: `fail – NagError *` **Input/Output**
   The NAG error parameter (see the Essential Introduction).

5 **Error Indicators and Warnings**

**NE_INT_ARG_LT**
   On entry, `n` must not be less than 3: `n = <value>`.

**NE_REAL_ARG_LT**
   On entry, `rho` must not be less than 0.0: `rho = <value>`.

**NE_BAD_PARAM**
   On entry, parameter `mode` had an illegal value.

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

**NE_NOT_STRICTLY_INCREASING**
   The sequence $x$ is not strictly increasing: $x[<value>] = <value>$, $x[<value>] = <value>$.

**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 consult NAG for assistance.

6 **Further Comments**
   The time taken by the routine is of order $n$. 

[NP3491/6]  

`g10abc.3`
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)`.

### 6.1 Accuracy

Accuracy 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.

### 6.2 References


### 7 See Also

- `nag_smooth_spline_estim (g10acc)`
- `nag_order_data (g10zac)`
- `nag_1d_spline_fit_knots (e02bac)`
- `nag_1d_spline_fit (e02bec)`

### 8 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 is fitted by `nag_smooth_spline_fit` with \(\rho = 10.0\). The fitted values and residuals are printed.

#### 8.1 Program Text

```c
/* nag_smooth_spline_fit (g10abc) Example Program. */
/* Copyright 2000 Numerical Algorithms Group. */
/* Mark 6, 2000. */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <naggl10.h>

int main (void)
{
  char mode[2], weight[2];
  double *coeff=0, df, *h=0, *res=0, rho, rss, *comm_ar=0, *weights=0, *wtptr,
  *wstart=0;
  double *x=0, *xord=0, *y=0, *yhat=0, *yord=0;
  Integer i, n, nord;
  Integer exit_status=0;
  NagError fail;
  Nag_SmoothFitType mode_enum;
```

`}
INIT_FAIL(fail);
Vprintf("gl0abc Example Program Results\n");

/* Skip heading in data file */
Vscanf("%*[\n]");

Vscanf("%d", &n);
if (!((coef = 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))
    || !(wtt = NAG_ALLOC(n, double))
    || !(yhat = NAG_ALLOC(n, double))
    || !(comm_ar = NAG_ALLOC(9*n+14, double)))
{
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

Vscanf("%s %s ", mode, weight);
if (*mode == 'P')
    mode_enum = Nag_SmoothFitPartial;
else if (*mode == 'Q')
    mode_enum = Nag_SmoothFitQuick;
else if (*mode == 'F')
    mode_enum = Nag_SmoothFitFull;
else
    mode_enum = (Nag_SmoothFitType)-999;

Vscanf("%lf", &rho);
if (*weight == 'U')
{
    for (i = 1; i <= n; ++i)
    Vscanf("%lf %lf ", &x[i - 1], &y[i - 1]);
    wptr = 0;
}
else
{
    for (i = 1; i <= n; ++i)
    Vscanf("%lf %lf ", &x[i - 1], &y[i - 1], &weights[i - 1]);
    wptr = weights;
}
/* Sort data into increasing X and */
/* remove tied observations and weight accordingly */
gl0zac(n, x, y, wptr, &nord, xord, yord, wtt, &rss, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from gl0zac.\n\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Fit cubic spline */
gl0abc

/* Print results */
Vprintf("\n");
Vprintf("%s10.3f\n", " rho = ", rho);
Vprintf("\n");
Vprintf("%s10.3f\n", " Residual sum of squares = ", rss);
Vprintf("%s10.3f\n", " Degrees of freedom = ", df);
Vprintf("\n");
Vprintf("%s\n", " Ordered input data Output results");
Vprintf("\n");
Vprintf("%s\n", " X Y Fitted Values");
Vprintf("\n");
for (i = 1; i <= nord; ++i)
{
    Vprintf("%8.4f %8.4f %8.4f\n",
            xord[i - 1],
            yord[i - 1],
            yhat[i - 1]);
}
END:
if (coeff) NAG_FREE(coeff);
if (h) NAG_FREE(h);
if (res) NAG_FREE(res);
if (x) NAG_FREE(x);
if (y) NAG_FREE(y);
if (weights) NAG_FREE(weights);
if (xord) NAG_FREE(xord);
if (yord) NAG_FREE(yord);
if (wwt) NAG_FREE(wwt);
if (yhat) NAG_FREE(yhat);
if (comm_ar) NAG_FREE(comm_ar);
return exit_status;
}

8.2 Program Data

gl0abc Example Program Data
43
F U
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

gl0abc.6
### 8.3 Program Results

gl0abc Example Program Results

\[ \text{rho} = 10.000 \]

Residual sum of squares = 11.288
Degrees of freedom = 27.785

<table>
<thead>
<tr>
<th>Ordered input data</th>
<th>Output results</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X )</td>
<td>( Y )</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0.9000</td>
<td>3.0000</td>
</tr>
<tr>
<td>1.0000</td>
<td>3.9000</td>
</tr>
<tr>
<td>1.8000</td>
<td>3.4000</td>
</tr>
<tr>
<td>1.9000</td>
<td>3.7000</td>
</tr>
<tr>
<td>2.2000</td>
<td>3.9000</td>
</tr>
<tr>
<td>4.2000</td>
<td>5.1000</td>
</tr>
<tr>
<td>4.8000</td>
<td>4.2000</td>
</tr>
<tr>
<td>5.1000</td>
<td>4.6000</td>
</tr>
<tr>
<td>5.2000</td>
<td>4.8500</td>
</tr>
<tr>
<td>5.8000</td>
<td>5.6000</td>
</tr>
<tr>
<td>6.9000</td>
<td>5.1000</td>
</tr>
<tr>
<td>7.9000</td>
<td>4.8000</td>
</tr>
<tr>
<td>8.1000</td>
<td>5.2000</td>
</tr>
<tr>
<td>8.5000</td>
<td>5.3000</td>
</tr>
<tr>
<td>8.8000</td>
<td>4.1000</td>
</tr>
<tr>
<td>8.9000</td>
<td>4.9000</td>
</tr>
<tr>
<td>9.8000</td>
<td>4.8000</td>
</tr>
<tr>
<td>9.9000</td>
<td>4.9000</td>
</tr>
<tr>
<td>10.4000</td>
<td>5.0000</td>
</tr>
<tr>
<td>10.5000</td>
<td>5.2000</td>
</tr>
<tr>
<td>10.6000</td>
<td>5.0000</td>
</tr>
<tr>
<td>10.8000</td>
<td>5.1000</td>
</tr>
<tr>
<td>11.0000</td>
<td>4.4000</td>
</tr>
<tr>
<td>11.1000</td>
<td>4.9000</td>
</tr>
<tr>
<td>11.3000</td>
<td>5.1000</td>
</tr>
<tr>
<td>11.5000</td>
<td>5.5000</td>
</tr>
<tr>
<td>11.8000</td>
<td>4.6000</td>
</tr>
<tr>
<td>11.9000</td>
<td>5.1000</td>
</tr>
<tr>
<td>12.4000</td>
<td>5.2000</td>
</tr>
<tr>
<td>12.5000</td>
<td>4.1000</td>
</tr>
<tr>
<td>12.7000</td>
<td>3.4000</td>
</tr>
<tr>
<td>12.8000</td>
<td>6.6000</td>
</tr>
<tr>
<td>13.8000</td>
<td>3.7000</td>
</tr>
<tr>
<td>14.5000</td>
<td>5.7000</td>
</tr>
<tr>
<td>15.5000</td>
<td>4.9000</td>
</tr>
<tr>
<td>15.6000</td>
<td>4.9000</td>
</tr>
</tbody>
</table>