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 <nagg10.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 ith observation and ρ 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 ρ weights these two aspects; larger values of ρ 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:

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

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

The parameter ρ can be chosen in a number of ways. The fit can be inspected for a number of different values of ρ. Alternatively the degrees of freedom for the spline, which determines the value of ρ, can be specified, or the (generalised) cross-validation can be minimised to give ρ; 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 ρ.

(3) Compute spline coefficients.

When fitting the spline for several different values of ρ, phase (1) need only be carried out once and then phase (2) repeated for different values of ρ. 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**[] ≥ 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, ρ.

Constraint: **rho**[] ≥ 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 ((**coeff**[][$(i-1)\times(n-1)+2$]$\times d$ + **coeff**[][$(i-1)\times(n-1)+1$])$\times d$ + **coeff**[][$(i-1)\times(n-1)$])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 $i = 1, 2, \ldots, n$.

13: **comm_ar[9*n+14]** – double *Input/Output*

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.

6.1 Accuracy

Accuracy depends on the value of ρ and the position of the x values. The values of $x_i - x_{i-1}$ and w_i are scaled and ρ is transformed to avoid underflow and overflow problems.

6.2 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

7 See Also

nag_smooth_spline_estim (g10acc)
nag_order_data (g10zac)

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

```
/* 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 <nagg10.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,
*wwt=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("g10abc Example Program Results\n");

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

  Vscanf("%ld", &n);
  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)))
    {
      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]);
      wtptr = 0;
    }
  else
    {
      for (i = 1; i <= n; ++i)
 Vscanf("%lf %lf %lf", &x[i - 1], &y[i - 1], &weights[i - 1]);
      wtptr = weights;
    }
  /*  Sort data into increasing X and */
  /*  remove tied observations and weight accordingly */
  g10zac(n, x, y, wtptr, &nord, xord, yord, wwt, &rss,
  &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from g10zac.\n%s\n", fail.message);
      exit_status = 1;
```

```
      goto END;
    }

  /*  Fit cubic spline */
  g10abc(mode_enum, nord, xord, yord, wwt, rho, yhat, coeff,
  &rss, &df, res, h, comm_ar, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from g10abc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }


  /*  Print results */
  Vprintf("\n");
  Vprintf("%s%10.3f\n", " rho  = ", rho);
  Vprintf("\n");
  Vprintf("%s%10.3f\n", " Residual sum of squares  = ", rss);
  Vprintf("%s%10.3f\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

```
g10abc 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
```

8.3 Program Results

```
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
  7.9000   4.8000        4.9590
  8.1000   5.2000        4.9262
  8.5000   5.3000        4.8595
  8.8000   4.1000        4.8172
  8.9000   4.9000        4.8095
  9.8000   4.8000        4.8676
  9.9000   4.9000        4.8818
 10.4000   5.0000        4.9445
 10.5000   5.2000        4.9521
 10.6000   5.0000        4.9572
 10.8000   5.1000        4.9613
 11.0000   4.4000        4.9614
 11.1000   4.9000        4.9618
 11.3000   5.1000        4.9623
 11.5000   5.5000        4.9568
 11.8000   4.6000        4.9338
 11.9000   5.1000        4.9251
 12.4000   5.2000        4.8943
 12.5000   4.1000        4.8944
 12.7000   3.4000        4.9051
 12.8000   6.6000        4.9138
 13.2000   5.3000        4.9239
 13.8000   3.7000        4.8930
 14.5000   5.7000        4.9938
 15.5000   4.9000        4.9773
 15.6000   4.9000        4.9682
```