1 Purpose

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

2 Specification

#include <nag.h>
#include <nagl0.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 non-linear 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 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 fitted see Hutchinson and de Hoog (1985) and Reinsch (1967).

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

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

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

The parameter $\rho$ 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. Minimise the cross-validation (CV), i.e., find $\rho$ such that the CV is minimised, where

$$
\text{CV} = \frac{1}{n} \sum_{i=1}^{n} \frac{r_i}{\sum_{j=1}^{n} \frac{r_j}{1 - h_{ij}}},
$$
(3) Minimise the generalised cross-validation (GCV), i.e., find \( \rho \) such that the GCV is minimised, where

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

\texttt{nag\_smooth\_spline\_estim} 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 \( \bar{y}_i \) 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 \texttt{nag\_order\_data} (g10zac).

The algorithm is based on Hutchinson (1986).

4 Parameters

1: \texttt{method} – Nag_SmoothParamMethods \hspace{1cm} \textit{Input}

\textit{On entry:} indicates whether the smoothing parameter is to be found by minimization of the CV or GCV functions, or by finding the smoothing parameter corresponding to a specified degrees of freedom value.

If \texttt{method} = \texttt{Nag\_SmoothParamCV}, cross-validation is used.

If \texttt{method} = \texttt{Nag\_SmoothParamDF}, the degrees of freedom are specified.

If \texttt{method} = \texttt{Nag\_SmoothParamGCV}, generalized cross-validation is used.

\textit{Constraint:} \texttt{method} = \texttt{Nag\_SmoothParamCV}, \texttt{Nag\_SmoothParamDF} or \texttt{Nag\_SmoothParamGCV}.

2: \texttt{n} – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} the number of observations, \( n \).

\textit{Constraint:} \( n \geq 3 \).

3: \texttt{x[n]} – const double \hspace{1cm} \textit{Input}

\textit{On entry:} the distinct and ordered values \( x_i \), for \( i = 1, 2, \ldots, n \).

\textit{Constraint:} \( x[i-1] < x[i] \), for \( i = 1, 2, \ldots, n-1 \).

4: \texttt{y[n]} – const double \hspace{1cm} \textit{Input}

\textit{On entry:} the values \( y_i \), for \( i = 1, 2, \ldots, n \).

5: \texttt{weights[n]} – const double \hspace{1cm} \textit{Input}

\textit{On entry:} \texttt{weights} must contain the \( n \) weights, if they are required. Otherwise, \texttt{weights} must be set to the null pointer (double*) \( 0 \).

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

6: \texttt{yhat[n]} – double \hspace{1cm} \textit{Output}

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

7: \texttt{coeff[n-1]*3]} – double \hspace{1cm} \textit{Output}

\textit{On exit:} the spline coefficients. More precisely, the value of the spline approximation at \( t \) is given by \((\text{coeff}[(i-1)\times(n-1)+2] \times d + \text{coeff}[(i-1)\times(n-1)+1] \times d + \text{coeff}[(i-1)\times(n-1)]) \times d + \hat{y}_i\), where \( x_i \leq t < x_{i+1} \) and \( d = t - x_i \).
g10 – Smoothing in Statistics

8: \( \text{rss} \) – double *

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

9: \( \text{df} \) – double *

On exit: the residual degrees of freedom. If \textbf{method} = \texttt{Nag_SmoothParamDF}, this will be \( n - \text{crit} \) to the required accuracy.

10: \( \text{res}[n] \) – double

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

11: \( \text{h}[n] \) – double

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

12: \( \text{crit} \) – double *

\textbf{Input/Output}

\textit{On entry:} if \textbf{method} = \texttt{Nag_SmoothParamDF}, the required degrees of freedom for the spline. If \textbf{method} = \texttt{Nag_SmoothParamCV} or \texttt{Nag_SmoothParamGCV}, \text{crit} need not be set.

\textit{Constraint:} \( 2.0 < \text{crit} \leq n \).

\textit{On exit:} if \textbf{method} = \texttt{Nag_SmoothParamCV}, the value of the cross-validation, or if \textbf{method} = \texttt{Nag_SmoothParamGCV}, the value of the generalized cross-validation function, evaluated at the value of \( \rho \) returned in \text{rho}.

13: \( \text{rho} \) – double *

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

14: \( \text{u} \) – double

\textbf{Input}

\textit{On entry:} the upper bound on the smoothing parameter. See Section 6 for details on how this parameter is used.

\textit{Constraint:} \( \text{u} > \text{tol} \).

\textit{Suggested value:} \( \text{u} = 1000.0 \).

15: \( \text{tol} \) – double

\textbf{Input}

\textit{On entry:} the accuracy to which the smoothing parameter \text{rho} is required. \text{tol} should be preferably not much less than \( \sqrt{\epsilon} \), where \( \epsilon \) is the \textit{machine precision}.

\textit{Constraint:} \( \text{tol} \geq \text{machine precision} \).

16: \( \text{maxcal} \) – Integer

\textbf{Input}

\textit{On entry:} the maximum number of spline evaluations to be used in finding the value of \( \rho \).

\textit{Constraint:} \( \text{maxcal} \geq 3 \).

\textit{Suggested value:} \( \text{maxcal} = 30 \).

17: \text{fail} – NagError *

\textbf{Input/Output}

The NAG error parameter (see the Essential Introduction).

5 Error Indicators and Warnings

\texttt{NE_INT_ARG_LT}

\textit{On entry,} \( n \) must not be less than 3: \( n = \textless \texttt{value} \).

\textit{On entry,} \( \texttt{maxcal} \) must not be less than 3: \( \texttt{maxcal} = \textless \texttt{value} \).
NE_BAD_PARAM
   On entry, parameter method had an illegal value.

NE_REAL
   On entry, crit = <value>.
   Constraint: crit > 2, if method = Nag_Smooth_Param_DF.

NE_REAL_INT_ARG_CONS
   On entry, crit = <value> and n = <value>.
   These parameters must satisfy crit \leq n, if method = Nag_Smooth_Param_DF.

NE_REAL_MACH_PREC
   On entry, tol = <value>, machine precision(X02AJC) = <value>.
   Constraint: tol \geq machine precision

NE_2_REAL_ARG_LE
   On entry, u = <value> while tol = <value>.
   These parameters must satisfy u > tol.

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_G10AC_DF_RHO
   method = Nag_Smooth_Param_DF and the required value of rho for specified degrees of freedom > u. Try a larger value of u.

NE_G10AC_DF_TOL
   method = Nag_Smooth_Param_DF and the accuracy given by tol cannot be achieved. Try increasing the value of tol.

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 = NagSmoothParamCV or Nag_SmoothParamGCV and the optimal value of rho > u. Try a larger value of u.

NE_ALLOC_FAIL
   Memory allocation failed.

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 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 minimised 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).

6.1 Accuracy

When minimising the cross-validation or generalised cross-validation, the error in the estimate of \( \rho \) should be within \( \pm 3 \times (\text{tol} \times \text{rho} + \text{tol}) \). When finding \( \rho \) for a fixed number of degrees of freedom the error in the estimate of \( \rho \) should be within \( \pm 2 \times \text{tol} \times \max(1, \text{rho}) \).

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.

6.2 References


7 See Also

nag_1d_spline_fit_knots (e02bac)
nag_1d_spline_fit (e02bec)
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 with 5 degrees of freedom is fitted by nag_smooth_spline_estim. The fitted values and residuals are printed.

8.1 Program Text

/* nag_smooth_spline_estim (g10acc) Example Program.  *
 * Copyright 2000 Numerical Algorithms Group.  *
 * Mark 6, 2000.  */

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

int main (void)
{
    char method[2], weight[2];
    double *coeff=0, crit, df, *h=0, *res=0, rho, rss, tol, u, *weights=0, *wtptr;
    double *wwt=0, *x=0, *xord=0, *y=0, *yhat=0, *yord=0;
    Integer i, maxcal, n, nord;
    Integer exit_status=0;
    NagError fail;
    Nag_SmoothParamMethods method_enum;

    INIT_FAIL(fail);
    Vprintf("gl0acc Example Program Results\n");

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

    Vscanf("%ld", &n);
    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))))
    
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

Vscanf(" %s %s ", method, weight);

if (*method == 'C')
    method_enum = Nag_SmoothParamCV;
else if (*method == 'G')
    method_enum = Nag_SmoothParamGCV;
else if (*method == 'D')
    method_enum = Nag_SmoothParamDF;
else
    method_enum = (Nag_SmoothParamMethods)-999;

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;
}
Vscanf("%lf %lf %ld %lf", &u, &tol, &maxcal, &crit);
/* Sort data, removing ties and weighting 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 */
g10acc(method_enum, nord, xord, yord, wwt, yhat, coeff, &rss, &df, res, h, &crit, &rho, u, tol, maxcal, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from g10acc\.n%s\n", fail.message);
  exit_status = 1;
goto END;
}

/* Print results */
Vprintf("\n");
Vprintf("%s\%10.2f\n", " Residual sum of squares = ", rss);
Vprintf("%s\%10.2f\n", " Degrees of freedom = ", df);
Vprintf("%s\%10.2f\n", " rho = ", rho);
Vprintf("\n");
Vprintf("\n\n%s%s%s%s%s\n", " Input data",
  " I X Y ",
  "YHAT H");
for (i = 1; i <= nord; ++i)
  Vprintf("%4ld %8.3f %8.3f %8.3f %8.3f\n",
    i, xord[i - 1], yord[i - 1], yhat[i - 1], h[i - 1]);
END:
  if (x) NAG_FREE(x);
  if (y) NAG_FREE(y);
  if (weights) NAG_FREE(weights);
  if (yhat) NAG_FREE(yhat);
  if (coeff) NAG_FREE(coeff);
  if (res) NAG_FREE(res);
  if (h) NAG_FREE(h);
  if (wwt) NAG_FREE(wwt);
  if (yord) NAG_FREE(yord);
  if (xord) NAG_FREE(xord);
  return exit_status;
}

8.2 Program Data

g10acc Example Program Data
43
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

[NP3491/6] g10acc.7
8.3 Program Results

g10acc Example Program Results

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

<table>
<thead>
<tr>
<th>Input data</th>
<th>Output results</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0.900</td>
</tr>
<tr>
<td>2</td>
<td>1.000</td>
</tr>
<tr>
<td>3</td>
<td>1.800</td>
</tr>
<tr>
<td>4</td>
<td>1.900</td>
</tr>
<tr>
<td>5</td>
<td>2.200</td>
</tr>
<tr>
<td>6</td>
<td>4.200</td>
</tr>
<tr>
<td>7</td>
<td>4.800</td>
</tr>
<tr>
<td>8</td>
<td>5.100</td>
</tr>
<tr>
<td>9</td>
<td>5.200</td>
</tr>
<tr>
<td>10</td>
<td>5.800</td>
</tr>
<tr>
<td>11</td>
<td>6.900</td>
</tr>
<tr>
<td>12</td>
<td>7.900</td>
</tr>
<tr>
<td>13</td>
<td>8.100</td>
</tr>
<tr>
<td>14</td>
<td>8.500</td>
</tr>
<tr>
<td>15</td>
<td>8.800</td>
</tr>
<tr>
<td>16</td>
<td>8.900</td>
</tr>
<tr>
<td>17</td>
<td>9.800</td>
</tr>
<tr>
<td>18</td>
<td>9.900</td>
</tr>
<tr>
<td>19</td>
<td>10.400</td>
</tr>
<tr>
<td>20</td>
<td>10.500</td>
</tr>
<tr>
<td>21</td>
<td>10.600</td>
</tr>
<tr>
<td>22</td>
<td>10.800</td>
</tr>
<tr>
<td>23</td>
<td>11.000</td>
</tr>
<tr>
<td>24</td>
<td>11.100</td>
</tr>
<tr>
<td>25</td>
<td>11.300</td>
</tr>
<tr>
<td>26</td>
<td>11.500</td>
</tr>
<tr>
<td>27</td>
<td>11.800</td>
</tr>
<tr>
<td>28</td>
<td>11.900</td>
</tr>
<tr>
<td>29</td>
<td>12.400</td>
</tr>
<tr>
<td>30</td>
<td>12.500</td>
</tr>
<tr>
<td>31</td>
<td>12.700</td>
</tr>
<tr>
<td>32</td>
<td>12.800</td>
</tr>
<tr>
<td>33</td>
<td>13.200</td>
</tr>
<tr>
<td>34</td>
<td>13.800</td>
</tr>
<tr>
<td>35</td>
<td>14.500</td>
</tr>
<tr>
<td>36</td>
<td>15.500</td>
</tr>
<tr>
<td>37</td>
<td>15.600</td>
</tr>
</tbody>
</table>