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

```c
#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, \( \hat{y} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n)^T \), 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 \( \text{trace}(I - H) \) 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 \( \text{trace}(H) = v_0 \) for given \( v_0 \).
2. Minimize the cross-validation (CV), i.e., find \( \rho \) such that the CV is minimized, where

\[
\text{CV} = \frac{1}{n} \sum_{i=1}^{n} \left( r_i \right)^2 \sum_{j=1}^{n} \left( 1 - h_{ij} \right).
\]
3. Minimize the generalized cross-validation (GCV), i.e., find \( \rho \) such that the GCV is minimized, where
\[ \text{GCV} = \frac{n^2}{\sum_{i=1}^{n} w_i} \left[ \frac{\sum_{i=1}^{n} r_i^2}{\left( \sum_{i=1}^{n} (1 - h_{ii}) \right)^2} \right]. \]

`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


## 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: \texttt{weights[n]} \texttt{— 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 \texttt{NULL}.

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

6: \texttt{yhat[n]} \texttt{— 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) \times 3]} \texttt{— double} \hspace{1cm} \textit{Output}

\textit{On exit:} the spline coefficients. More precisely, the value of the spline approximation at \(t\) is given by

\[
((\texttt{coeff[(i - 1) \times (n - 1) + 2] \times d + coeff[(i - 1) \times (n - 1) + 1]}) \times d + \texttt{yhat[i]}), \text{ where } x_i \leq t < x_{i+1} \text{ and } d = t - x_i.
\]

8: \texttt{rss} \texttt{— double *} \hspace{1cm} \textit{Output}

\textit{On exit:} the (weighted) residual sum of squares.

9: \texttt{df} \texttt{— double *} \hspace{1cm} \textit{Output}

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

10: \texttt{res[n]} \texttt{— double} \hspace{1cm} \textit{Output}

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

11: \texttt{h[n]} \texttt{— double} \hspace{1cm} \textit{Output}

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

12: \texttt{crit} \texttt{— double *} \hspace{1cm} \textit{Input/Output}

\textit{On entry:} if \texttt{method = Nag_SmoothParamDF}, the required degrees of freedom for the spline.

If \texttt{method = Nag_SmoothParamCV} or \texttt{Nag_SmoothParamGCV}, \texttt{crit} need not be set.

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

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

13: \texttt{rho} \texttt{— double *} \hspace{1cm} \textit{Output}

\textit{On exit:} the smoothing argument, \(\rho\).

14: \texttt{u} \texttt{— double} \hspace{1cm} \textit{Input}

\textit{On entry:} the upper bound on the smoothing argument. See Section 9 for details on how this argument is used.

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

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

15: \texttt{tol} \texttt{— double} \hspace{1cm} \textit{Input}

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

\textit{Constraint:} \(\texttt{tol} \geq \text{machine precision}\).
16: \textbf{maxcal} – Integer

*Input*

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

\textit{Suggested value:} \textbf{maxcal} = 30.

\textit{Constraint:} \textbf{maxcal} \geq 3.

17: \textbf{fail} – \text{NagError} *

*Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

\textbf{NE\_2\_REAL\_ARG\_LE}

On entry, \( u = \langle \text{value} \rangle \) while \( \text{tol} = \langle \text{value} \rangle \). These arguments must satisfy \( u > \text{tol} \).

\textbf{NE\_ALLOC\_FAIL}

Dynamic memory allocation failed.

\textbf{NE\_BAD\_PARAM}

On entry, argument \textit{method} had an illegal value.

\textbf{NE\_G10\_AC\_ACC}

A solution to the accuracy given by \textit{tol} has not been achieved in \textbf{maxcal} iterations. Try increasing the value of \textit{tol} and/or \textbf{maxcal}.

\textbf{NE\_G10\_AC\_CG\_RHO}

\textit{method} = \text{Nag\_SmoothParamCV} or \text{Nag\_SmoothParamGCV} and the optimal value of \textit{rho} > \( u \).

Try a larger value of \( u \).

\textbf{NE\_G10\_AC\_DF\_RHO}

\textit{method} = \text{Nag\_SmoothParamDF} and the required value of \textit{rho} for specified degrees of freedom > \( u \).

Try a larger value of \( u \).

\textbf{NE\_G10\_AC\_DF\_TOL}

\textit{method} = \text{Nag\_SmoothParamDF} and the accuracy given by \textit{tol} cannot be achieved. Try increasing the value of \textit{tol}.

\textbf{NE\_INT\_ARG\_LT}

On entry, \textbf{maxcal} = \langle \text{value} \rangle.

\textit{Constraint:} \textbf{maxcal} \geq 3.

On entry, \textbf{n} = \langle \text{value} \rangle.

\textit{Constraint:} \textbf{n} \geq 3.

\textbf{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.

\textbf{NE\_NOT\_STRICLY\_INCREASING}

The sequence \( x \) is not strictly increasing: \( x[\langle \text{value} \rangle] = \langle \text{value} \rangle \), \( x[\langle \text{value} \rangle] = \langle \text{value} \rangle \).
NE_REAL
On entry, crit = ⟨value⟩.
Constraint: crit > 2.0, if method = Nag_SmoothParamDF.

NE_REAL_ARRAY_CONS
On entry, weights[⟨value⟩] = ⟨value⟩.
Constraint: weights[i] > 0, for i = 0, 1, ..., n − 1.

NE_REAL_INT_ARG_CONS
On entry, crit = ⟨value⟩ and n = ⟨value⟩. These arguments must satisfy crit ≤ n, if method = Nag_SmoothParamDF.

NE_REAL_MACH_PREC
On entry, tol = ⟨value⟩, machine precision (nag_machine_precision) = ⟨value⟩.
Constraint: tol ≥ machine precision.

7 Accuracy
When minimizing the cross-validation or generalized cross-validation, the error in the estimate of ρ should be within ±3 × (tol × ρ + tol). When finding ρ for a fixed number of degrees of freedom the error in the estimate of ρ should be within ±2 × tol × max(1, ρ).

Given the value of ρ, the accuracy of the fitted spline depends on the value of ρ and the position of the x values. The values of 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 to fit the spline for a given value of ρ is of order n.

When finding the value of ρ that gives the required degrees of freedom, the algorithm examines the interval 0.0 to u. For small degrees of freedom the value of ρ can be large, as in the theoretical case of two degrees of freedom when the spline reduces to a straight line and ρ 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, xi, and C-peptide concentration (pmol/ml), yi, 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.
#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, *wt.ptr;
  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("%39s", nag_enum_arg, _countof(nag_enum_arg));
  #else
    scanf("%39s", nag_enum_arg);
  #endif

  method = (Nag_SmoothParamMethods) nag_enum_name_to_value(nag_enum_arg);

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

  PRINT_RESULT(fail, method, x, y, n, nord, res, crit, df, rho, rss, tol, u, coeff, h, wwt, yhat, yord, xord, weights);

  return exit_status;
}

END:
exit_status = -1;
return;

// End nag_smooth_spline_estim (g10acc).
#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
    wtpr = 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
    wtpr = 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 */
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 */
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("%s\n", " Input data",
" i x y ",
" yhat h ");
for (i = 1; i <= nord; ++i)
    printf("%s\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(coef);
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

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

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
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>12.400</td>
<td>5.200</td>
<td>4.852</td>
<td>0.206</td>
</tr>
<tr>
<td>30</td>
<td>12.500</td>
<td>4.100</td>
<td>4.857</td>
<td>0.196</td>
</tr>
<tr>
<td>31</td>
<td>12.700</td>
<td>3.400</td>
<td>4.900</td>
<td>0.189</td>
</tr>
<tr>
<td>32</td>
<td>12.800</td>
<td>6.600</td>
<td>4.932</td>
<td>0.193</td>
</tr>
<tr>
<td>33</td>
<td>13.200</td>
<td>5.300</td>
<td>4.955</td>
<td>0.488</td>
</tr>
<tr>
<td>34</td>
<td>13.800</td>
<td>3.700</td>
<td>4.797</td>
<td>0.408</td>
</tr>
<tr>
<td>35</td>
<td>14.500</td>
<td>5.700</td>
<td>5.076</td>
<td>0.559</td>
</tr>
<tr>
<td>36</td>
<td>15.500</td>
<td>4.900</td>
<td>4.979</td>
<td>0.445</td>
</tr>
<tr>
<td>37</td>
<td>15.600</td>
<td>4.900</td>
<td>4.946</td>
<td>0.535</td>
</tr>
</tbody>
</table>