NAG Library Function Document

nag_2d_triang_interp (e01sjc)

1 Purpose

nag_2d_triang_interp (e01sjc) generates a two-dimensional surface interpolating a set of scattered data points, using the method of Renka and Cline.

2 Specification

```c
#include <nag.h>
#include <nage01.h>
void nag_2d_triang_interp (Integer m, const double x[], const double y[],
        const double f[], Integer triang[], double grads[], NagError *fail)
```

3 Description

nag_2d_triang_interp (e01sjc) constructs an interpolating surface \( F(x,y) \) through a set of \( m \) scattered data points \((x_r, y_r, f_r)\), for \( r = 1, 2, \ldots, m \), using a method due to Renka and Cline. In the \((x,y)\) plane, the data points must be distinct. The constructed surface is continuous and has continuous first derivatives.

The method involves firstly creating a triangulation with all the \((x,y)\) data points as nodes, the triangulation being as nearly equiangular as possible (see Cline and Renka (1984)). Then gradients in the \(x\)- and \(y\)-directions at node \( r \), for \( r = 1, 2, \ldots, m \), are estimated using a method due to Renka and Cline. In the \((x,y)\) plane, the data points must be distinct. The constructed surface is continuous and has continuous first derivatives.

The method involves firstly creating a triangulation with all the \((x,y)\) data points as nodes, the triangulation being as nearly equiangular as possible (see Cline and Renka (1984)). Then gradients in the \(x\)- and \(y\)-directions at node \( r \), for \( r = 1, 2, \ldots, m \), are estimated using a method due to Renka and Cline. In the \((x,y)\) plane, the data points must be distinct. The constructed surface is continuous and has continuous first derivatives.

The computed partial derivatives, with the \( f_r \) values, at the three nodes of each triangle define a piecewise polynomial surface of a certain form which is the interpolant on that triangle. See Renka and Cline (1984) for more detailed information on the algorithm, a development of that by Lawson (1977). The code is derived from Renka (1984).

The interpolant \( F(x,y) \) can subsequently be evaluated at any point \((x,y)\) inside or outside the domain of the data by a call to nag_2d_triang_eval (e01skc). Points outside the domain are evaluated by extrapolation.

4 References


5 Arguments

1: \( m \) – Integer \hspace{1cm} \textit{Input}

\textit{On entry:} \( m \), the number of data points.

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

2: \( x[m] \) – const double \hspace{1cm} \textit{Input}

3: \( y[m] \) – const double \hspace{1cm} \textit{Input}

4: \( f[m] \) – const double \hspace{1cm} \textit{Input}

\textit{On entry:} the coordinates of the \( r \)th data point, for \( r = 1, 2, \ldots, m \). The data points are accepted in any order, but see Section 9.

\textit{Constraint:} the \((x, y)\) nodes must not all be collinear, and each node must be unique.

5: \( \text{triang}[7 \times m] \) – Integer \hspace{1cm} \textit{Output}

\textit{On exit:} a data structure defining the computed triangulation, in a form suitable for passing to \text{nag_2d_triang_eval (e01skc)}.

6: \( \text{grads}[2 \times m] \) – double \hspace{1cm} \textit{Output}

\textit{Note:} the \((i, j)\)th element of the matrix is stored in \( \text{grads}[(j - 1) \times 2 + i - 1] \).

\textit{On exit:} the estimated partial derivatives at the nodes, in a form suitable for passing to \text{nag_2d_triang_eval (e01skc)} \( \text{grads}[(r - 1) \times 2] \) and \( \text{grads}[(r - 1) \times 2 + 1] \) respectively, for \( r = 1, 2, \ldots, m \).

7: \( \text{fail} \) – NagError \* \hspace{1cm} \textit{Input/Output}

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

6 Error Indicators and Warnings

\textbf{NE_ALL_DATA_COLLINEAR}

All nodes are collinear. There is no unique solution.

\textbf{NE_ALLOC_FAIL}

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE_BAD_PARAM}

On entry, argument \langle value \rangle had an illegal value.

\textbf{NE_DATA_NOT_UNIQUE}

On entry, \((x[I - 1], y[I - 1]) = (x[J - 1], y[J - 1])\), for \( I, J = \langle value \rangle \langle value \rangle \), \( x[I - 1] \), \( y[I - 1] = \langle value \rangle \langle value \rangle \).

\textbf{NE_INT}

On entry, \( m = \langle value \rangle \).

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

\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.
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.

7 Accuracy
On successful exit, the computational errors should be negligible in most situations but you should always check the computed surface for acceptability, by drawing contours for instance. The surface always interpolates the input data exactly.

8 Parallelism and Performance
Not applicable.

9 Further Comments
The time taken for a call of nag_2d_triang_interp (e01sjc) is approximately proportional to the number of data points, \(m\). The function is more efficient if, before entry, the values in \(x\), \(y\) and \(f\) are arranged so that the \(x\) array is in ascending order.

10 Example
This example reads in a set of 30 data points and calls nag_2d_triang_interp (e01sjc) to construct an interpolating surface. It then calls nag_2d_triang_eval (e01skc) to evaluate the interpolant at a sample of points on a rectangular grid.

Note that this example is not typical of a realistic problem: the number of data points would normally be larger, and the interpolant would need to be evaluated on a finer grid to obtain an accurate plot, say.

10.1 Program Text

```c
/* nag_2d_triang_interp (e01sjc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 8, 2004. */
*/
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage01.h>

int main(void)
{

    /* Scalars */
    double xhi, xlo, yhi, ylo;
    Integer exit_status, i, j, m, nx, ny;

    /* Arrays */
    double *f = 0, *grads = 0, *pf = 0, *px = 0, *py = 0, *x = 0, *y = 0;
    Integer *triang = 0;

    /* Nag Types */
    NagError fail;

    exit_status = 0;
    INIT_FAIL(fail);
```
printf("nag_2d_triang_interp (e01sjc) Example Program Results\n");

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

/* Input the number of nodes. */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[\n ] ", &m);
#else
  scanf("%"NAG_IFMT"%*[\n ] ", &m);
#endif
if (m >= 3)
{
  /* Allocate memory */
  if (!(f = NAG_ALLOC(m, double)) ||
      !(grads = NAG_ALLOC(2 * m, double)) ||
      !(x = NAG_ALLOC(m, double)) ||
      !(y = NAG_ALLOC(m, double)) ||
      !(triang = NAG_ALLOC(7*m, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
}
else
{
  printf("Invalid m.\n");
  exit_status = 1;
  goto END;
}

/* Input the nodes (X,Y) and heights, F. */
for (i = 1; i <= m; ++i)
{
#ifdef _WIN32
  scanf_s("%lf%lf%lf%*[\n ] ", &x[i - 1], &y[i - 1], &f[i - 1]);
#else
  scanf("%lf%lf%lf%*[\n ] ", &x[i - 1], &y[i - 1], &f[i - 1]);
#endif

/* Generate the triangulation and gradients. */

/* nag_2d_triang_interp (e01sjc).
   * A function to generate a two-dimensional surface
   * interpolating a set of data points, using either the
   * method of Renka and Cline or the modified Shepard’s
   * method
   */
  nag_2d_triang_interp(m, x, y, f, triang, grads, &fail);
  if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_2d_triang_interp (e01sjc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Evaluate the interpolant on a rectangular grid at NX*NY points */
/* over the domain (XLO to XHI) x (YLO to YHI). */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%lf%lf%*[\n ] ", &nx, &xlo, &xhi);
#else
  scanf("%"NAG_IFMT"%lf%lf%*[\n ] ", &nx, &xlo, &xhi);
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%lf%lf%*[\n ] ", &ny, &ylo, &yhi);
#else
  scanf("%"NAG_IFMT"%lf%lf%*[\n ] ", &ny, &ylo, &yhi);
#endif
#else
    scanf("%NAG_IFMT%lf%lf%*[\n] ", &ny, &ylo, &yhi);
#endif

if (nx > 0 && ny > 0)
{
    /* Allocate memory */
    if (!(pf = NAG_ALLOC(nx, double)) ||
        !(px = NAG_ALLOC(nx, double)) ||
        !(py = NAG_ALLOC(ny, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else
{
    printf("Invalid nx or ny.\n");
    exit_status = 1;
    goto END;
}

for (i = 1; i <= nx; ++i)
{
    px[i - 1] = (double)(nx - i) / (nx - 1) * xlo +
                (double)(i - 1) / (nx - 1) * xhi;
}
for (i = 1; i <= ny; ++i)
{
    py[i - 1] = (double)(ny - i) / (ny - 1) * ylo +
                (double)(i - 1) / (ny - 1) * yhi;
}
printf("\n");
printf("%s", " X");
for (i = 1; i <= nx; ++i)
{
    printf("%8.2f", px[i - 1]);
}
printf("\n");
printf("%s", " Y");
printf("\n");
for (i = ny; i >= 1; --i)
{
    for (j = 1; j <= nx; ++j)
    {
        /* nag_2d_triang_eval (e01skc).
           * A function to evaluate, at a set of points, the
           * two-dimensional interpolant function generated by
           * nag_2d_triang_interp (e01sjc).
           */
        nag_2d_triang_eval(m, x, y, f, triang, grads, px[j - 1],
                         py[i - 1], &pf[j - 1], &fail);
        if (fail.code != NE_NOERROR)
        {
            printf("Error from nag_2d_triang_eval (e01skc).\n",
                    fail.message);
            exit_status = 1;
            goto END;
        }
    }
    printf("%8.2f", py[i - 1]);
    printf("%3s", "");
    for (j = 1; j <= nx; ++j)
    {
        printf("%8.2f", pf[j - 1]);
    }
    printf("\n");
}
END:
NAG_FREE(f);
NAG_FREE(grads);
NAG_FREE(pf);
NAG_FREE(px);
NAG_FREE(py);
NAG_FREE(x);
NAG_FREE(y);
NAG_FREE(triang);

return exit_status;
}

10.2 Program Data

nag_2d_triang_interp (e01sjc) Example Program Data
30 M, the number of data points
11.16 1.24 22.15 X, Y, F data point definition
12.85 3.06 22.11
19.85 10.72 7.97
19.72 1.39 16.83
15.91 7.74 15.30
0.00 20.00 34.60
20.87 20.00 5.74
3.45 12.78 41.24
14.26 17.87 10.74
17.43 3.46 18.60
22.80 12.39 5.47
7.58 1.98 29.87
25.00 11.87 4.40
9.66 20.00 4.73
5.22 14.66 40.36
17.25 19.57 6.43
25.00 3.87 8.74
12.13 10.79 13.71
22.23 6.21 10.25
11.52 8.53 15.74
15.20 0.00 21.60
7.54 10.69 19.31
17.32 13.78 12.11
2.14 15.03 53.10
0.51 8.37 49.43
22.69 19.63 3.25
5.47 17.13 28.63
21.67 14.36 5.52
3.31 0.33 44.08 End of the data points
7 3.0 21.0 Grid definition, X axis
6 2.0 17.0 Grid definition, Y axis

10.3 Program Results

nag_2d_triang_interp (e01sjc) Example Program Results

<table>
<thead>
<tr>
<th>X</th>
<th>3.00</th>
<th>6.00</th>
<th>9.00</th>
<th>12.00</th>
<th>15.00</th>
<th>18.00</th>
<th>21.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>17.00</td>
<td>41.25</td>
<td>27.62</td>
<td>18.03</td>
<td>12.29</td>
<td>11.68</td>
<td>9.09</td>
</tr>
<tr>
<td></td>
<td>14.00</td>
<td>47.61</td>
<td>36.66</td>
<td>22.87</td>
<td>14.02</td>
<td>13.44</td>
<td>11.20</td>
</tr>
<tr>
<td></td>
<td>11.00</td>
<td>38.55</td>
<td>25.25</td>
<td>16.72</td>
<td>13.83</td>
<td>13.08</td>
<td>10.71</td>
</tr>
<tr>
<td></td>
<td>8.00</td>
<td>37.90</td>
<td>23.97</td>
<td>16.79</td>
<td>16.43</td>
<td>15.46</td>
<td>13.02</td>
</tr>
<tr>
<td></td>
<td>5.00</td>
<td>40.49</td>
<td>29.26</td>
<td>22.51</td>
<td>20.72</td>
<td>19.30</td>
<td>16.72</td>
</tr>
<tr>
<td></td>
<td>2.00</td>
<td>43.52</td>
<td>33.91</td>
<td>26.59</td>
<td>22.23</td>
<td>21.15</td>
<td>18.67</td>
</tr>
</tbody>
</table>