NAG Library Function Document

nag_2d_scat_interpolant (e01sac)

1 Purpose

nag_2d_scat_interpolant (e01sac) generates a two-dimensional surface interpolating a set of scattered data points, using either the method of Renka and Cline or a modification of Shepard's method.

2 Specification

```
#include <nag.h>
#include <nage01.h>
```

```
void nag_2d_scat_interpolant (Nag_2d_Scat_Method method, Integer m,
     const double x[], const double y[], const double f[], Nag_Scat_Struct *comm,
     Nag_E01_Opt *optional, NagError *fail)
```

3 Description

nag_2d_scat_interpolant (e01sac) 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 either a method due to Renka and Cline or a modification of Shepard's method. In the (x,y) plane, the data points must be distinct. The constructed surface is continuous and has continuous first derivatives.

If **method** $=$ Nag_RC, the method due to Renka and Cline is used. This 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 are estimated at node r, for $r = 1, 2, \ldots, m$, as the partial derivatives of a quadratic function of x and y which interpolates the data value f_r, and which fits the data values at nearby nodes (those within a certain distance chosen by the algorithm) in a weighted least-squares sense. The weights are chosen such that closer nodes have more influence than more distant nodes on derivative estimates at node r. 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 n points (x_k, y_k), for $k = 1, 2, \ldots, n$, inside or outside the domain of the data by a call to nag_2d_scat_eval (e01sbc). Points outside the domain are evaluated by extrapolation.

If **method** $=$ Nag_Shep, then a modification of Shepard's method is used. The basic Shepard method, described in Shepard (1968), interpolates the input data with the weighted mean

$$F(x,y) = \frac{\sum_{r=1}^{m} w_r(x,y) f_r}{\sum_{r=1}^{m} w_r(x,y)},$$

$$\text{where } w_r(x,y) = \frac{1}{d_r^2} \text{ and } d_r^2 = (x - x_r)^2 + (y - y_r)^2.$$

The basic method is global in that the interpolated value at any point depends on all the data, but this function uses a modification due to Franke and Nielson (1980), whereby the method becomes local by adjusting each $w_r(x,y)$ to be zero outside a circle with centre (x_r, y_r) and some radius R_w. Also, to improve the performance of the basic method, each f_r above is replaced by a function $f_r(x,y)$, which is a quadratic fitted by weighted least-squares to data local to (x_r, y_r) and forced to interpolate (x_r, y_r, f_r). In this context, a point (x,y) is defined to be local to another point if it lies within some distance R_q of it. Computation of these quadratics constitutes the main work done by this part of the function. If there are less than 5 other points within distance R_q from (x_r, y_r), the quadratic is replaced by a linear function. In cases of rank-deficiency, the minimum norm solution is computed.

You may specify values for R_w and R_q, but it is usually easier to choose instead two integers N_w and N_q, from which the function will compute R_w and R_q. These integers can be thought of as the average numbers of data points lying within distances R_w and R_q respectively from each node. Default values are provided, and advice on alternatives is given in Section 8.

The interpolant $F(x, y)$ generated by this function can subsequently be evaluated for n points (x_k, y_k), for $k = 1, 2, \ldots, n$, in the domain of the data by a call to nag_2d_scat_eval (e01sbc).

4 References

Cline A K and Renka R L (1984) A storage-efficient method for construction of a Thiessen triangulation *Rocky Mountain J. Math.* **14** 119–139

Franke R and Nielson G (1980) Smooth interpolation of large sets of scattered data *Internat. J. Num. Methods Engrg.* **15** 1691–1704

Lawson C L (1977) Software for C^1 surface interpolation *Mathematical Software III* (ed J R Rice) 161–194 Academic Press

Renka R L (1984) Algorithm 624: Triangulation and interpolation of arbitrarily distributed points in the plane *ACM Trans. Math. Software* **10** 440–442

Renka R L and Cline A K (1984) A triangle-based C^1 interpolation method *Rocky Mountain J. Math.* **14** 223–237

Shepard D (1968) A two-dimensional interpolation function for irregularly spaced data *Proc. 23rd Nat. Conf. ACM* 517–523 Brandon/Systems Press Inc., Princeton

5 Arguments

1: **method** – Nag_2d_Scat_Method *Input*

On entry: indicates the method to be used in interpolating the surface.

method = Nag_RC
The method due to Renka and Cline is used.

method = Nag_Shep
A modification of Shepard's method is used.

Constraint: **method** = Nag_RC or Nag_Shep.

2: **m** – Integer *Input*

On entry: the number of data points, m.

Constraint: **m** ≥ 3.

3: **x**[**m**] – const double *Input*
4: **y**[**m**] – const double *Input*
5: **f**[**m**] – const double *Input*

On entry: the co-ordinates of the rth data point, for $r = 1, 2, \ldots, m$. The data points are accepted in any order, but see Section 8.

Constraint: Each node must be unique. Additionally for **method** = Nag_RC the (x, y) nodes must not be all collinear.

6: **comm** – Nag_Scat_Struct * *Output*

Pointer to a communication structure of type Nag_Scat_Struct. For **method** = Nag_RC, this structure contains the computed triangulation and the estimated partial derivatives at the nodes. For **method** = Nag_Shep, this structure contains the coefficients of the constructed nodal functions.

This structure must be passed **unchanged** to the interpolant evaluating function nag_2d_scat_eval (e01sbc).

7: **optional** – Nag_E01_Opt *

Pointer to structure of type Nag_E01_Opt, which may be used only if **method** = Nag_Shep. It has the following members:

rnw – double *Input/Output*
rnq – double *Input/Output*

On entry: suitable values for the radii R_w and R_q, described in Section 3.

Constraints:

$0 <$ **rnw** $\leq$ **rnq**;
If this constraint is satisfied then these values are used, otherwise if **rnq** ≤ 0 then suitable values of R_w and R_q are computed from other members of this structure, **nw** and **nq**.

On exit: **rnw** and **rnq** contain the actual values of R_w and R_q used by nag_2d_scat_interpolant (e01sac).

nw – Integer *Input/Output*
nq – Integer *Input/Output*

On entry: these are integer values which can be used to compute the radii R_w and R_q, if you do not wish to supply appropriate values directly, i.e., **rnq** ≤ 0.

Constraints:

$0 <$ **nw** $\leq$ **nq**;
If this constraint is satisfied then nag_2d_scat_interpolant (e01sac) computes the appropriate values of the radii R_w and R_q. Otherwise, if **optional** $\rightarrow$ **nq** ≤ 0, then **nw** and **nq** are set to 9 and 18 respectively and these values are used to compute the radii R_w and R_q.

On exit: if the values of **rnw** and **rnq** as supplied are used for the radii R_w and R_q then **nw** and **nq** are set to zero, otherwise **nw** and **nq** contain the actual values used for computing the radii R_w and R_q.

If this structure is supplied as a null pointer (Nag_E01_Opt) $\times$ 0, also defined by Nag as `E01_DEFAULT` when **method** = Nag_Shep, then the default values of **nw** = 9 and **nq** = 18 are used to compute the radii.

minnq – Integer *Output*

On exit: the minimum number of data points that lie within radius **rnq** of any node, and thus define a nodal function. If **minnq** is very small (say, less than 5), then the interpolant may be unsatisfactory in regions where the data points are sparse. nag_2d_scat_interpolant (e01sac) generates a warning in this case.

Note that this structure is not referenced when **method** = Nag_RC. It should however be specified as `E01_DEFAULT`.

8: **fail** – NagError * *Input/Output*

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

6 Error Indicators and Warnings

NE_ALL_DATA_COLLINEAR

On entry, all the (**x**,**y**) pairs are collinear. Consider specifying **method** = Nag_Shep or using a one-dimensional interpolating function nag_1d_spline_interpolant (e01bac).

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument **method** had an illegal value.

NE_DATA_NOT_UNIQUE

On entry, each data pair is not unique since data points $(\mathbf{x}[\langle value\rangle], \mathbf{y}[\langle value\rangle])$ and $(\mathbf{x}[\langle value\rangle], \mathbf{y}[\langle value\rangle])$ are identical and equal to $(\langle value\rangle, \langle value\rangle)$.

NE_INT_ARG_LT

On entry, $\mathbf{m} = \langle value\rangle$.
Constraint: $\mathbf{m} \geq 3$.

NE_NUM_PARAM_INVALID

On entry, either or both of **optional** → **nq** and **optional** → **nw** are invalid, **optional** → **nq** = ⟨*value*⟩ and **optional** → **nw** = ⟨*value*⟩. **optional** → **nq** and **optional** → **nw** must satisfy the following constraints: 0 < **optional** → **nw** ≤ **optional** → **nq**.

NE_RAD_PARAM_INVALID

On entry, either or both of **optional** → **rnq** and **optional** → **rnw** are invalid, **optional** → **rnq** = ⟨*value*⟩ and **optional** → **rnw** = ⟨*value*⟩. **optional** → **rnq** and **optional** → **rnw** must satisfy the following constraints: 0.0 < **optional** → **rnw** ≤ **optional** → **rnq**.

NW_SPARSE_DATA_FIT

The minimum number of data points ⟨*value*⟩ that lie within the radius **optional** → **rnq** of any node is small enough to indicate that the interpolant may be unsatisfactory in regions where the data points are sparse. Current values of other relevant arguments (available as members of the structure **optional**, if this has been defined) are **optional** → **rnq** = ⟨*value*⟩, **optional** → **rnw** = ⟨*value*⟩, **optional** → **nq** = ⟨*value*⟩, **optional** → **nw** = ⟨*value*⟩.

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 Further Comments

If **method** = Nag_RC, the time taken for a call of nag_2d_scat_interpolant (e01sac) is approximately proportional to the number of data points, m. The function is more efficient if, before entry, the values in **x**, **y**, **f** are arranged so that the **x** array is in ascending order.

If **method** = Nag_Shep, the time taken for a call of nag_2d_scat_interpolant (e01sac) is also approximately proportional to the number of data points, m, provided that N_q is of the same order as its default value (18). However, if N_q is increased so that the method becomes more global, the time taken becomes approximately proportional to m^2.

In particular, when **method** = Nag_Shep, note first that the radii R_w and R_q, described in Section 3, are computed as $\frac{D}{2}\sqrt{\frac{N_w}{m}}$ and $\frac{D}{2}\sqrt{\frac{N_q}{m}}$ respectively, where D is the maximum distance between any pair of data points.

Default values $N_w = 9$ and $N_q = 18$ work quite well when the data points are fairly uniformly distributed. However, for data having some regions with relatively few points or for small datasets ($m < 25$), a larger value of N_w may be needed. This is to ensure a reasonable number of data points within a distance R_w of each node, and to avoid some regions in the data area being left outside all the discs of radius R_w on which the weights $w_r(x,y)$ are nonzero. Maintaining N_q approximately equal to $2N_w$ is usually an advantage.

Note however that increasing N_w and N_q does not improve the quality of the interpolant in all cases. It does increase the computational cost and makes the method less local.

At the end of the program, especially before any repeated call to nag_2d_scat_interpolant (e01sac), the function nag_2d_scat_free (e01szc) must be called with the communication structure **comm** as the argument in order to free the memory previously allocated to the pointer members of **comm**.

9 Example

This program reads in a set of 30 data points and calls nag_2d_scat_interpolant (e01sac) to construct an interpolating surface. It then calls nag_2d_scat_eval (e01sbc) to evaluate the interpolant at a sample of points on a rectangular grid. The two methods described in Section 3 are used in the construction of the interpolating surface and the subsequent evaluation of the interpolant.

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.

9.1 Program Text

```
/* nag_2d_scat_interpolant (e01sac) Example Program.
 *
 * Copyright 1996 Numerical Algorithms Group.
 *
 * Mark 4, 1996.
 * Mark 8 revised, 2004.
 */

#include <nag.h>
#include <nagx04.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nage01.h>

int main(int argc, char *argv[])
{
  FILE               *fpin, *fpout;

  Integer            exit_status = 0, i, isel, j, m, n, nx, ny;
  NagError           fail;
  Nag_2d_Scat_Method method;
  Nag_E01_Opt        optional;
  Nag_Scat_Struct    comm;
  double             *f = 0, *pf = 0, *px = 0, *py = 0, *x = 0, xhi, xlo;
  double             *y = 0, yhi, ylo;

  INIT_FAIL(fail);

  /* Check for command-line IO options */
  fpin = nag_example_file_io(argc, argv, "-data", NULL);
  fpout = nag_example_file_io(argc, argv, "-results", NULL);
  fprintf(fpout, "nag_2d_scat_interpolant (e01sac) Example Program Results\n");
  /* Skip heading in data file */
  fscanf(fpin, "%*[^\n]");
  /* Input the number of nodes. */
  fscanf(fpin, "%ld", &m);

  if (m >= 3)
    {
      if (!(f = NAG_ALLOC(m, double)) ||
          !(x = NAG_ALLOC(m, double)) ||
          !(y = NAG_ALLOC(m, double)))
        {
          fprintf(fpout, "Allocation failure\n");
          exit_status = -1;
          goto ENDL;
        }
    }
  else
    {
      fprintf(fpout, "Invalid m.\n");
      exit_status = 1;
      return exit_status;
    }
  /* Input the nodes (x,y) and heights, f. */
  for (i = 0; i < m; ++i)
    fscanf(fpin, "%lf%lf%lf", &x[i], &y[i], &f[i]);

  for (isel = 1; isel <= 2; ++isel)
    {
      /* Select the method of interpolation. */
      if (isel == 1)
        method = Nag_RC;
      else
        method = Nag_Shep;

      if (method == Nag_RC)
        {
          fprintf(fpout, "\nExample 1: Surface interpolation by method "
                  "of Renka and Cline.\n\n");
          /*
           * Generate the triangulation and gradients using the selected
           * method.
           */

          /* nag_2d_scat_interpolant (e01sac).
           * 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_scat_interpolant(method, m, x, y, f, &comm, (Nag_E01_Opt *) 0,
                                  &fail);
          if (fail.code != NE_NOERROR)
            {
              fprintf(fpout,
                      "Error from nag_2d_scat_interpolant (e01sac).\n%s\n",
                      fail.message);
              exit_status = 1;
              goto END;
            }
        }
      else if (method == Nag_Shep)
        {
          fprintf(fpout, "\n\nExample 2: Surface interpolation by modified "
                  "Shepard's method.\n\n");
          /* Compute the nodal function coefficients. */
          optional.nq = 24;
          optional.nw = 12;
          optional.rnq = -1.0;

          /* nag_2d_scat_interpolant (e01sac), see above. */
          nag_2d_scat_interpolant(method, m, x, y, f, &comm, &optional, &fail);
          if (fail.code != NE_NOERROR)
            {
              fprintf(fpout,
                      "Error from nag_2d_scat_interpolant (e01sac).\n%s\n",
                      fail.message);
              exit_status = 1;
              goto END;
            }

          fprintf(fpout, "   optional.rnw =%8.2f  optional.rnq =%8.2f\n\n",
                  optional.rnw, optional.rnq);
          fprintf(fpout, "   minimum number of data points that lie within "
                  "radius optional.rnq =%3ld\n", optional.minnq);
        }
      /* Input the domain for evaluating the interpolant. */
      fscanf(fpin, "%ld%lf%lf", &nx, &xlo, &xhi);
      fscanf(fpin, "%ld%lf%lf", &ny, &ylo, &yhi);

      if (nx >= 1 && ny >= 1)
        {
          if (!(pf = NAG_ALLOC(nx*ny, double)) ||
              !(px = NAG_ALLOC(nx*ny, double)) ||
              !(py = NAG_ALLOC(nx*ny, double)))
            {
              fprintf(fpout, "Allocation failure\n");
              exit_status = -1;
              goto END;
            }
        }
      else
        {
          fprintf(fpout, "Invalid nx or ny.\n");
          exit_status = 1;
          return exit_status;
        }
      /*
       * Evaluate the interpolant on a rectangular grid at nx*ny points
       * over the domain (xlo to xhi) x (ylo to yhi).
       */
      n = 0;
      for (j = 0; j < ny; ++j)
        {
          for (i = 0; i < nx; ++i)
            {
              px[i+nx*j] = ((double)(nx-i-1) / (nx-1)) * xlo +
                           ((double)(i) / (nx-1)) * xhi;
              py[i+nx*j] = ((double)(ny-j-1) / (ny-1)) * ylo +
                           ((double)(j) / (ny-1)) * yhi;
              ++n;
            }
        }
      if (method == Nag_RC)
        {
          /* nag_2d_scat_eval (e01sbc).
           * A function to evaluate, at a set of points, the
           * two-dimensional interpolant function generated by
           * nag_2d_scat_interpolant (e01sac)
           */
          nag_2d_scat_eval(&comm, n, px, py, pf, &fail);
          if (fail.code != NE_NOERROR)
            {
              fprintf(fpout, "Error from nag_2d_scat_eval (e01sbc).\n%s\n",
                      fail.message);
              exit_status = 1;
              goto END;
            }
        }
      else if (method == Nag_Shep)
        {
          /* nag_2d_scat_eval (e01sbc), see above. */
          nag_2d_scat_eval(&comm, n, px, py, pf, &fail);
        }
      fprintf(fpout, "\n          x");
      for (i = 0; i < nx; i++)
        fprintf(fpout, "%8.2f", px[i]);
      fprintf(fpout, "\n     y\n");
      for (i = ny-1; i >= 0; --i)
        {
          fprintf(fpout, "%8.2f   ", py[nx * i]);
          for (j = 0; j < nx; j++)
            fprintf(fpout, "%8.2f", pf[nx * i + j]);
          fprintf(fpout, "\n");
```

```
        }
 END:
      /* Free the memory allocated to the pointers in structure comm. */
      /* nag_2d_scat_free (e01szc).
       * Freeing function for use with nag_2d_scat_eval (e01sbc)
       */
      nag_2d_scat_free(&comm);
      if (pf) NAG_FREE(pf);
      if (px) NAG_FREE(px);
      if (py) NAG_FREE(py);
    }
 ENDL:
  if (fpin != stdin) fclose(fpin);
  if (fpout != stdout) fclose(fpout);
  if (f) NAG_FREE(f);
  if (x) NAG_FREE(x);
  if (y) NAG_FREE(y);
  return exit_status;
}
```

9.2 Program Data

```
nag_2d_scat_interpolant (e01sac) Example Program Data
      30
     11.16    1.24   22.15
     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
      0.00    0.00   58.20
      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
   7    3.0   21.0
   6    2.0   17.0
```

9.3 Program Results

```
nag_2d_scat_interpolant (e01sac) Example Program Results

Example 1: Surface interpolation by method of Renka and Cline.


          x     3.00    6.00    9.00   12.00   15.00   18.00   21.00
     y
   17.00       41.25   27.62   18.03   12.29   11.68    9.09    5.37
   14.00       47.61   36.66   22.87   14.02   13.44   11.20    6.46
   11.00       38.55   25.25   16.72   13.83   13.08   10.71    6.88
    8.00       37.90   23.97   16.79   16.43   15.46   13.02    9.30
    5.00      40.49   29.26   22.51   20.72   19.30   16.72   12.87
    2.00      43.52   33.91   26.59   22.23   21.15   18.67   14.88

Example 2: Surface interpolation by modified Shepard's method.

   optional.rnw =    9.49  optional.rnq =   13.42

   minimum number of data points that lie within radius optional.rnq =  7

          x    3.00    6.00    9.00   12.00   15.00   18.00   21.00
     y
   17.00      40.23   27.72   21.23   14.59   12.00    9.43    5.46
   14.00      46.96   37.37   23.74   14.67   13.25   11.29    6.26
   11.00      39.42   25.42   16.32   13.78   12.60   10.39    7.03
    8.00      37.50   22.36   18.57   15.63   15.55   13.05    9.69
    5.00      41.25   31.76   24.74   21.17   18.93   16.83   12.65
    2.00      44.58   34.35   26.47   22.27   20.98   18.69   15.06
```