# NAG C Library Function Document

# nag_3d_shep_interp (e01tgc)

## 1 Purpose

nag_3d_shep_interp (e01tgc) generates a three-dimensional interpolant to a set of scattered data points, using a modified Shepard method.

## 2 Specification

```
void nag_3d_shep_interp (Integer m, const double x[], const double y[],
     const double z[], const double f[], Integer nw, Integer nq, Integer iq[],
     double rq[], NagError *fail)
```

## 3 Description

This function constructs a smooth function $Q(x, y, z)$ which interpolates a set of $m$ scattered data points $(x_r, y_r, z_r, f_r)$, for $r = 1, 2, \ldots, m$, using a modification of Shepard's method. The surface is continuous and has continuous first partial derivatives.

The basic Shepard method, which is a generalization of the two-dimensional method described in Shepard (1968), interpolates the input data with the weighted mean

$$Q(x, y, z) = \frac{\sum_{r=1}^{m} w_r(x, y, z) q_r}{\sum_{r=1}^{m} w_r(x, y, z)},$$

where

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

The basic method is global in that the interpolated value at any point depends on all the data, but this function uses a modification (see Franke and Nielson (1980), Renka (1988c)), whereby the method becomes local by adjusting each $w_r(x, y, z)$ to be zero outside a sphere with centre $(x_r, y_r, z_r)$ and some radius $R_w$. Also, to improve the performance of the basic method, each $q_r$ above is replaced by a function $q_r(x, y, z)$, which is a quadratic fitted by weighted least-squares to data local to $(x_r, y_r, z_r)$ and forced to interpolate $(x_r, y_r, z_r, f_r)$. In this context, a point $(x, y, z)$ 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 function.

The efficiency of the function is further enhanced by using a cell method for nearest neighbour searching due to Bentley and Friedman (1979).

The radii $R_w$ and $R_q$ are chosen to be just large enough to include $N_w$ and $N_q$ data points, respectively, for user-supplied constants $N_w$ and $N_q$. Default values of these parameters are provided by the function, and advice on alternatives is given in Section 8.2.

This function is derived from the function QSHEP3 described by Renka (1988b).

Values of the interpolant $Q(x, y, z)$ generated by this function, and its first partial derivatives, can subsequently be evaluated for points in the domain of the data by a call to nag_3d_shep_eval (e01thc).

## 4 References

Bentley J L and Friedman J H (1979) Data structures for range searching *ACM Comput. Surv.* **11** 397–409

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

Renka R J (1988c) Multivariate interpolation of large sets of scattered data *ACM Trans. Math. Software* **14** 139–148

Renka R J (1988b) Algorithm 661: QSHEP3D: Quadratic Shepard method for trivariate interpolation of scattered data *ACM Trans. Math. Software* **14** 151–152

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 Parameters

1: **m** – Integer *Input*

*On entry*: $m$, the number of data points.

*Constraint*: **m** $\geq 10$.

2: **x**[**m**] – const double *Input*
3: **y**[**m**] – const double *Input*
4: **z**[**m**] – const double *Input*

*On entry*: **x**$[r-1]$, **y**$[r-1]$, **z**$[r-1]$ must be set to the Cartesian coordinates of the data point $(x_r, y_r, z_r)$, for $r = 1, 2, \ldots, m$.

*Constraint*: these coordinates must be distinct, and must not all be coplanar.

5: **f**[**m**] – const double *Input*

*On entry*: **f**$[r-1]$ must be set to the data value $f_r$, for $r = 1, 2, \ldots, m$.

6: **nw** – Integer *Input*

*On entry*: the number $N_w$ of data points that determines each radius of influence $R_w$, appearing in the definition of each of the weights $w_r$, for $r = 1, 2, \ldots, m$ (see Section 3). Note that $R_w$ is different for each weight. If **nw** $\leq 0$ the default value **nw** $= \min(32, \mathbf{m} - 1)$ is used instead.

*Constraint*: **nw** $\leq \min(40, \mathbf{m} - 1)$.

7: **nq** – Integer *Input*

*On entry*: the number $N_q$ of data points to be used in the least-squares fit for coefficients defining the nodal functions $q_r(x, y, z)$ (see Section 3). If **nq** $\leq 0$ the default value **nq** $= \min(17, \mathbf{m} - 1)$ is used instead.

*Constraint*: **nq** $\leq 0$ or $9 \leq$ **nq** $\leq \min(40, \mathbf{m} - 1)$.

8: **iq**$[dim]$ – Integer *Output*

**Note:** the dimension, $dim$, of the array **iq** must be at least $2 \times$ **m** $+ 1$.

*On exit*: integer data defining the interpolant $Q(x, y, z)$.

9: **rq**$[dim]$ – double *Output*

**Note:** the dimension, $dim$, of the array **rq** must be at least $10 \times$ **m** $+ 7$.

*On exit*: real data defining the interpolant $Q(x, y, z)$.

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

The NAG error parameter (see the Essential Introduction).

# 6 Error Indicators and Warnings

**NE_INT**

On entry, **m** = ⟨*value*⟩.
Constraint: **m** $\geq 10$.

On entry, **nq** $> 0$ and **nq** $< 9$: **nq** = ⟨*value*⟩.

**NE_INT_2**

On entry, **nw** $> \min(40, \mathbf{m} - 1)$: **nw** = ⟨*value*⟩, **m** = ⟨*value*⟩.

On entry, **nq** $> \min(40, \mathbf{m} - 1)$: **nq** = ⟨*value*⟩, **m** = ⟨*value*⟩.

**NE_DATA_COPLANAR**

All nodes are coplanar. There is no unique solution.

**NE_DUPLICATE_NODE**

There are duplicate nodes in the data set. $(\mathbf{x}[i-1], \mathbf{y}[i-1], \mathbf{z}[i-1]) = (\mathbf{x}[j-1], \mathbf{y}[j-1], \mathbf{z}[j-1])$ for: $i =$ ⟨*value*⟩ and $j =$ ⟨*value*⟩. The interpolant cannot be derived.

**NE_BAD_PARAM**

On entry, parameter ⟨*value*⟩ had an illegal 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.

# 7 Accuracy

On successful exit, the function generated interpolates the input data exactly and has quadratic accuracy.

# 8 Further Comments

## 8.1 Timing

The time taken for a call to nag_3d_shep_interp (e01tgc) will depend in general on the distribution of the data points. If **x**, **y** and **z** are uniformly randomly distributed, then the time taken should be O(**m**). At worst O($\mathbf{m}^2$) time will be required.

## 8.2 Choice of $N_w$ and $N_q$

Default values of the parameters $N_w$ and $N_q$ may be selected by calling nag_3d_shep_interp (e01tgc) with **nw** $\leq 0$ and **nq** $\leq 0$. These default values may well be satisfactory for many applications.

If non-default values are required they must be supplied to nag_3d_shep_interp (e01tgc) through positive values of **nw** and **nq**. Increasing these parameters makes the method less local. This may increase the accuracy of the resulting interpolant at the expense of increased computational cost. The default values **nw** $= \min(32, \mathbf{m} - 1)$ and **nq** $= \min(17, \mathbf{m} - 1)$ have been chosen on the basis of experimental results reported in Renka (1988c). In these experiments the error norm was found to vary smoothly with $N_w$ and $N_q$, generally increasing monotonically and slowly with distance from the optimal pair. The method is not therefore thought to be particularly sensitive to the parameter values. For further advice on the choice of these parameters see Renka (1988c).

# 9 Example

This program reads in a set of 30 data points and calls nag_3d_shep_interp (e01tgc) to construct an interpolating function $Q(x, y, z)$. It then calls nag_3d_shep_eval (e01thc) to evaluate the interpolant at a set of points.

Note that this example is not typical of a realistic problem: the number of data points would normally be larger.

## 9.1 Program Text

```
/* nag_3d_shep_interp (e01tgc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

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

int main(void)
{
  /* Scalars */
  Integer exit_status, i, m, n, nq, nw, liq, lrq;
  NagError fail;

  /* Arrays */
  double *f = 0, *q = 0, *qx = 0, *qy = 0, *qz = 0, *rq = 0,
         *u = 0, *v = 0, *w = 0, *x = 0, *y = 0, *z = 0;
  Integer *iq = 0;

  exit_status = 0;

  INIT_FAIL(fail);
  Vprintf("e01tgc Example Program Results\n");

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

  /* Input the number of nodes. */
  Vscanf("%ld%*[^\n] ", &m);

  if (m > 0)
    {
      /* Allocate memory */
      lrq = 10 * m + 7;
      liq = 2 * m + 1;
      if ( !(f = NAG_ALLOC(m, double)) ||
           !(x = NAG_ALLOC(m, double)) ||
           !(y = NAG_ALLOC(m, double)) ||
           !(z = NAG_ALLOC(m, double)) ||
           !(rq = NAG_ALLOC(lrq, double)) ||
           !(iq = NAG_ALLOC(liq, Integer)) )
        {
          Vprintf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }

      /* Input the data points X,Y,Z and F. */
      for (i = 0; i < m; ++i)
        Vscanf("%lf%lf%lf%lf%*[^\n] ", &x[i], &y[i], &z[i], &f[i]);

      /* Generate the interpolant. */
      nq = 0;
      nw = 0;
```

```
      e01tgc(m, x, y, z, f, nw, nq, iq, rq, &fail);
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from e01tgc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }

      /* Input the number of evaluation points. */
      Vscanf("%ld%*[^\n] ", &n);

      /* Allocate memory for e01thc */
      if ( !(q = NAG_ALLOC(n, double)) ||
           !(qx = NAG_ALLOC(n, double)) ||
           !(qy = NAG_ALLOC(n, double)) ||
           !(qz = NAG_ALLOC(n, double)) ||
           !(u = NAG_ALLOC(n, double)) ||
           !(v = NAG_ALLOC(n, double)) ||
           !(w = NAG_ALLOC(n, double)) )
        {
          Vprintf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }

      /* Input the evaluation points. */
      for (i = 0; i < n; ++i)
        Vscanf("%lf%lf%lf%*[^\n] ", &u[i], &v[i], &w[i]);

      /* Evaluate the interpolant using e01thc. */
      fail.print = TRUE;
      e01thc(m, x, y, z, f, iq, rq, n, u, v, w, q, qx, qy, qz, &fail);

      Vprintf("\n");
      Vprintf("     i     u(i)      v(i)      w(i)      Q(i)\n");
      for (i = 0; i < n; ++i)
        Vprintf("%6ld%10.4f%10.4f%10.4f%10.4f\n", i, u[i], v[i], w[i], q[i]);
    }

 END:
  if (f) NAG_FREE(f);
  if (q) NAG_FREE(q);
  if (qx) NAG_FREE(qx);
  if (qy) NAG_FREE(qy);
  if (qz) NAG_FREE(qz);
  if (rq) NAG_FREE(rq);
  if (u) NAG_FREE(u);
  if (v) NAG_FREE(v);
  if (w) NAG_FREE(w);
  if (x) NAG_FREE(x);
  if (y) NAG_FREE(y);
  if (z) NAG_FREE(z);
  if (iq) NAG_FREE(iq);

  return exit_status;
}
```

## 9.2 Program Data

```
e01tgc Example Program Data
 30                    M, the number of data points
 0.80  0.23  0.37  0.51  X, Y, Z, F data point definition
 0.23  0.88  0.05  1.80
 0.18  0.43  0.04  0.11
 0.58  0.95  0.62  2.65
 0.64  0.69  0.20  0.93
 0.88  0.35  0.49  0.72
 0.30  0.10  0.78 -0.11
 0.87  0.09  0.05  0.67
```

```
 0.04  0.02  0.40  0.00
 0.62  0.90  0.43  2.20
 0.87  0.96  0.24  3.17
 0.62  0.64  0.45  0.74
 0.86  0.13  0.47  0.64
 0.87  0.60  0.46  1.07
 0.49  0.43  0.13  0.22
 0.12  0.61  0.00  0.41
 0.02  0.71  0.82  0.58
 0.62  0.93  0.44  2.48
 0.49  0.54  0.04  0.37
 0.36  0.56  0.39  0.35
 0.62  0.42  0.97 -0.20
 0.01  0.72  0.45  0.78
 0.41  0.36  0.52  0.11
 0.17  0.99  0.65  2.82
 0.51  0.29  0.59  0.14
 0.85  0.05  0.04  0.61
 0.20  0.20  0.87 -0.25
 0.04  0.67  0.04  0.59
 0.31  0.63  0.18  0.50
 0.88  0.27  0.07  0.71  End of data points
 6                       N, the number of evaluation points
 0.10  0.10  0.10        U, V, W evaluation point definition
 0.20  0.20  0.20
 0.30  0.30  0.30
 0.40  0.40  0.40
 0.50  0.50  0.50
 0.60  0.60  0.60        End of evaluation points
```

## 9.3 Program Results

```
e01tgc Example Program Results

     i     u(i)      v(i)      w(i)      Q(i)
     0    0.1000    0.1000    0.1000    0.2630
     1    0.2000    0.2000    0.2000    0.1182
     2    0.3000    0.3000    0.3000    0.0811
     3    0.4000    0.4000    0.4000    0.1552
     4    0.5000    0.5000    0.5000    0.3019
     5    0.6000    0.6000    0.6000    0.5712
```