# Module 8.4: nag_scat_interp
# Interpolation of Scattered Data

nag_scat_interp provides procedures for computing and evaluating interpolating functions through scattered data sets in two and three dimensions.

## Contents

# Procedure: nag_scat_2d_interp

## 1 Description

nag_scat_2d_interp generates a smooth function $q(x, y)$ which interpolates a set of $m$ scattered data points $(x_i, y_i, f_i)$, for $i = 1, 2, \ldots, m$, using a modified quadratic Shepard method. The function $q(x, y)$ is continuous and has continuous first partial derivatives.

Values of the interpolant generated by this procedure, and its first partial derivatives, can subsequently be evaluated for any point in the domain of the data by a call to nag_scat_2d_eval.

## 2 Usage

```
USE nag_scat_interp

CALL nag_scat_2d_interp(x, y, f, interp  [, optional arguments])
```

## 3 Arguments

**Note.** All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**($n$)' is used in the argument descriptions to specify that the array **x** must have exactly $n$ elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

$m \geq 6$ — the number of data points

### 3.1 Mandatory Arguments

**x**($m$) — real(kind=*wp*), intent(in)

**y**($m$) — real(kind=*wp*), intent(in)

*Input:* the values $x_i$ and $y_i$, for $i = 1, 2, \ldots, m$.

*Constraints:* the data points in the $(x, y)$ plane must be distinct ($(x_j, y_j) \neq (x_k, y_k)$, for $j \neq k$) and must not all be collinear.

**f**($m$) — real(kind=*wp*), intent(in)

*Input:* the values $f_i$ for $i = 1, 2, \ldots, m$.

**interp** — type(nag_scat_comm_*wp*), intent(out)

*Output:* a structure containing details of the interpolant $q(x, y)$ generated. This structure may be passed to the procedure nag_scat_2d_eval to evaluate $q(x, y)$, and optionally its first partial derivatives, at given points.

*Note:* to reduce the risk of corrupting the data accidentally, the components of this structure are private; details of $q(x, y)$ may be extracted by calling nag_scat_extract.

The procedure allocates roughly $9m + 5$ real(kind=*wp*) and $2m$ integer elements of storage to the structure. If you wish to deallocate this storage when the structure is no longer required, you must call the procedure nag_deallocate, as illustrated in Example 1 of this module document.

### 3.2 Optional Arguments

**Note.** Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

**nw** — integer, intent(in), optional

*Input:* the integer $N_w$ used in the modified Shepard method (see Section 6.1 for details).

*Default:* `nw` $= \min(19, m-1)$.

*Constraints:* $1 \leq$ `nw` $\leq \min(40, m-1)$.

**nq** — integer, intent(in), optional

*Input:* the integer $N_q$ used in the modified Shepard method (see Section 6.1 for details).

*Default:* `nq` $= \min(13, m-1)$.

*Constraints:* $5 \leq$ `nq` $\leq \min(40, m-1)$.

**error** — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

## 4 Error Codes

### Fatal errors (error%level = 3):

| error%code | Description |
|---|---|
| **301** | An input argument has an invalid value. |
| **302** | An array argument has an invalid shape. |
| **303** | Array arguments have inconsistent shapes. |
| **320** | The procedure was unable to allocate enough memory. |

## 5 Examples of Usage

Complete examples of the use of this procedure appear in Examples 1 and 3 of this module document.

## 6 Further Comments

### 6.1 Algorithmic Detail

The basic Shepard method, described in Shepard [6], interpolates the input data with the weighted mean

$$q(x,y) = \frac{\sum_{i=1}^{m} w_i(x,y) q_i}{\sum_{i=1}^{m} w_i(x,y)},$$

where

$$w_i(x,y) = \frac{1}{d_i^2} \text{ and } d_i^2 = (x-x_i)^2 + (y-y_i)^2.$$

The basic method is global in that the interpolated value at any point depends on all the data, but this procedure uses a modification (see Franke and Nielson [2] and Renka [3]), whereby the method becomes local by adjusting each $w_i(x,y)$ to be zero outside a circle with centre $(x_i, y_i)$ and some radius $R_w$. Also, to improve the performance of the basic method, each $q_i$ above is replaced by a function $q_i(x,y)$, which is a quadratic fitted by weighted least-squares to data local to $(x_i, y_i)$ and forced to interpolate $(x_i, y_i, f_i)$.

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 procedure.

The efficiency of the procedure is further enhanced by using a cell method for nearest neighbour searching due to Bentley and Friedman [1].

The radii $R_w$ and $R_q$ are chosen to be just large enough to include $N_w$ and $N_q$ data points, respectively. Non-default values of $N_w$ and $N_q$ may be set if required by supplying the optional arguments `nw` and/or `nq`. Increasing the values of these arguments 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` $= \min(19, m-1)$ and `nq` $= \min(13, m-1)$ have been chosen on the basis of experimental results reported in Renka [3]. 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 [3].

This procedure is derived from the routine QSHEP2 described in Renka [4].

## 6.2 Accuracy

On successful exit, the function generated interpolates the input data exactly and has quadratic accuracy (see Renka [3]).

## 6.3 Timing

The time taken for a call to this procedure will depend in general on the distribution of the data points. If `x` and `y` are approximately uniformly distributed, then the time taken should be $O(m)$. At worst $O(m^2)$ time will be required.

# Procedure: nag_scat_3d_interp

## 1 Description

nag_scat_3d_interp generates a smooth function $q(x, y, z)$ which interpolates a set of $m$ scattered data points $(x_i, y_i, z_i, f_i)$, for $i = 1, 2, \ldots, m$, using a modified quadratic Shepard method. The function $q(x, y, z)$ is continuous and has continuous first partial derivatives.

Values of the interpolant generated by this procedure, and its first partial derivatives, can subsequently be evaluated for any point in the domain of the data by a call to nag_scat_3d_eval.

## 2 Usage

```
USE nag_scat_interp

CALL nag_scat_3d_interp(x, y, z, f, interp  [, optional arguments])
```

## 3 Arguments

**Note.** All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly $n$ elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

$m \geq 10$ — the number of data points

### 3.1 Mandatory Arguments

**x**($m$) — real(kind=*wp*), intent(in)

**y**($m$) — real(kind=*wp*), intent(in)

**z**($m$) — real(kind=*wp*), intent(in)

*Input:* the values $x_i$, $y_i$ and $z_i$, for $i = 1, 2, \ldots, m$.

*Constraints:* the data points must be distinct ($(x_j, y_j, z_j) \neq (x_k, y_k, z_k)$, for $j \neq k$) and must not all be coplanar.

**f**($m$) — real(kind=*wp*), intent(in)

*Input:* the values $f_i$ for $i = 1, 2, \ldots, m$.

**interp** — type(nag_scat_comm_*wp*), intent(out)

*Output:* a structure containing details of the interpolant $q(x, y, z)$ generated. This structure may be passed to the procedure nag_scat_3d_eval to evaluate $q(x, y, z)$, and optionally its first partial derivatives, at given points.

*Note:* to reduce the risk of corrupting the data accidentally, the components of this structure are private; details of $q(x, y, z)$ may be extracted by calling nag_scat_extract.

The procedure allocates roughly $14m + 7$ real(kind=*wp*) and $2m$ integer elements of storage to the structure. If you wish to deallocate this storage when the structure is no longer required, you must call the procedure nag_deallocate, as illustrated in Example 2 of this module document.

### 3.2 Optional Arguments

**Note.** Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

**nw** — integer, intent(in), optional

*Input:* the integer $N_w$ used in the modified Shepard method (see Section 6.1 for details).

*Default:* `nw` $= \min(32, m-1)$.

*Constraints:* $1 \leq$ `nw` $\leq \min(40, m-1)$.

**nq** — integer, intent(in), optional

*Input:* the integer $N_q$ used in the modified Shepard method (see Section 6.1 for details).

*Default:* `nq` $= \min(17, m-1)$.

*Constraints:* $9 \leq$ `nq` $\leq \min(40, m-1)$.

**error** — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

## 4 Error Codes

### Fatal errors (error%level = 3):

| error%code | Description |
|---|---|
| **301** | An input argument has an invalid value. |
| **302** | An array argument has an invalid shape. |
| **303** | Array arguments have inconsistent shapes. |
| **320** | The procedure was unable to allocate enough memory. |

## 5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

## 6 Further Comments

### 6.1 Algorithmic Detail

The basic Shepard method, which is a 3-d generalization of the method described in Shepard [6], interpolates the input data with the weighted mean

$$q(x,y,z) = \frac{\sum_{i=1}^{m} w_i(x,y,z) q_i}{\sum_{i=1}^{m} w_i(x,y,z)},$$

where

$$w_i(x,y,z) = \frac{1}{d_i^2} \text{ and } d_i^2 = (x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2.$$

The basic method is global in that the interpolated value at any point depends on all the data, but this procedure uses a modification (see Franke and Nielson [2] and Renka [3]), whereby the method becomes local by adjusting each $w_i(x,y,z)$ to be zero outside a sphere with centre $(x_i, y_i, z_i)$ and some radius $R_w$. Also, to improve the performance of the basic method, each $q_i$ above is replaced by a function $q_i(x,y,z)$,

which is a quadratic fitted by weighted least-squares to data local to $(x_i, y_i, z_i)$ and forced to interpolate $(x_i, y_i, z_i, f_i)$. 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 procedure.

The efficiency of the procedure is further enhanced by using a cell method for nearest neighbour searching due to Bentley and Friedman [1].

The radii $R_w$ and $R_q$ are chosen to be just large enough to include $N_w$ and $N_q$ data points, respectively. Non-default values of $N_w$ and $N_q$ may be set if required by supplying the optional arguments `nw` and/or `nq`. Increasing the values of these arguments 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` $= \min(32, m-1)$ and `nq` $= \min(17, m-1)$ have been chosen on the basis of experimental results reported in Renka [3]. 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 [3].

This procedure is derived from the routine QSHEP3 described in Renka [5].

## 6.2 Accuracy

On successful exit, the function generated interpolates the input data exactly and has quadratic accuracy (see Renka [3]).

## 6.3 Timing

The time taken for a call to this procedure will depend in general on the distribution of the data points. If `x`, `y` and `z` are approximately uniformly distributed, then the time taken should be $O(m)$. At worst $O(m^2)$ time will be required.

# Procedure: nag_scat_2d_eval

## 1 Description

nag_scat_2d_eval computes values of a 2-d interpolant $q(x, y)$, and optionally its first partial derivatives, where $q(x, y)$ was generated by nag_scat_2d_interp.

This procedure is generic and may be used to evaluate the interpolant

- at a single point,
- at a set of $n$ scattered points,
- or on an $n_x \times n_y$ grid of points.

## 2 Usage

```
USE nag_scat_interp
CALL nag_scat_2d_eval(interp, u, v, q  [, optional arguments])
```

### 2.1 Interfaces

Distinct interfaces are provided for the following cases:

Evaluation at a single point / at scattered points / at points on a grid

| | |
|---|---|
| **Single point:** | u, v, q and the optional arguments qx and qy must all be scalars. |
| **Scattered points:** | u, v, q and the optional arguments qx and qy must all be rank-1 arrays. |
| **Points on a grid:** | u and v must be rank-1 arrays. q and the optional arguments qx and qy must be rank-2 arrays. |

## 3 Arguments

**Note.** All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly $n$ elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$n \geq 1$ — the total number of evaluation points (for scattered points)
$n_x \geq 1$ — the number of grid points in the $x$-direction (for points on a grid)
$n_y \geq 1$ — the number of grid points in the $y$-direction (for points on a grid)

### 3.1 Mandatory Arguments

**interp** — type(nag_scat_comm_*wp*), intent(in)
*Input:* a structure containing details of the interpolant $q(x, y)$ to be evaluated.
*Constraints:* interp must be as output from a previous call to nag_scat_2d_interp or nag_scat_2d_set.

**u / u($n$) / u($n_x$)** — real(kind=*wp*), intent(in)
*Input:* the $x$-coordinate(s) of the evaluation point(s).
*Constraints:*
If q is scalar, u must be scalar;
if q is a rank-1 array, u must also be a rank-1 array, with SIZE(u) = SIZE(q);
if q is a rank-2 array, u must be a rank-1 array, with SIZE(u) = SIZE(q,1).

**v / v($n$) / v($n_y$)** — real(kind=*wp*), intent(in)

*Input:* the $y$-coordinate(s) of the evaluation point(s).

*Constraints:*

If `q` is scalar, `v` must be scalar;

if `q` is a rank-1 array, `v` must also be a rank-1 array, with `SIZE(v)` = `SIZE(q)`;

if `q` is a rank-2 array, `v` must be a rank-1 array, with `SIZE(v)` = `SIZE(q,2)`.

**q / q($n$) / q($n_x, n_y$)** — real(kind=*wp*), intent(out)

*Output:* the value(s) of the interpolant at the evaluation point(s).

If `q` is a scalar (single evaluation point), `q` = $q(\mathtt{u}, \mathtt{v})$;

if `q` is a rank-1 array (scattered evaluation points), `q` = $q(\mathtt{u}(i), \mathtt{v}(i))$, for $i = 1, 2, \ldots, n$;

if `q` is a rank-2 array (evaluation on a grid), `q` = $q(\mathtt{u}(j), \mathtt{v}(k))$, for $j = 1, 2, \ldots, n_x$, $k = 1, 2, \ldots, n_y$.

## 3.2 Optional Arguments

**Note.** Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

**qx / qx($n$) / qx($n_x, n_y$)** — real(kind=*wp*), intent(out), optional

*Output:* the value(s) of the first partial derivative $q_x$ of the interpolant at the evaluation point(s).

If `qx` is a scalar (single evaluation point), `qx` = $q_x(\mathtt{u}, \mathtt{v})$;

if `qx` is a rank-1 array (scattered evaluation points), `qx` = $q_x(\mathtt{u}(i), \mathtt{v}(i))$, for $i = 1, 2, \ldots, n$;

if `qx` is a rank-2 array (evaluation on a grid), `qx` = $q_x(\mathtt{u}(j), \mathtt{v}(k))$, for $j = 1, 2, \ldots, n_x$, $k = 1, 2, \ldots, n_y$.

**qy / qy($n$) / qy($n_x, n_y$)** — real(kind=*wp*), intent(out), optional

*Output:* the value(s) of the first partial derivative $q_y$ of the interpolant at the evaluation point(s).

If `qy` is a scalar (single evaluation point), `qy` = $q_y(\mathtt{u}, \mathtt{v})$;

if `qy` is a rank-1 array (scattered evaluation points), `qy` = $q_y(\mathtt{u}(i), \mathtt{v}(i))$, for $i = 1, 2, \ldots, n$;

if `qy` is a rank-2 array (evaluation on a grid), `qy` = $q_y(\mathtt{u}(j), \mathtt{v}(k))$, for $j = 1, 2, \ldots, n_x$, $k = 1, 2, \ldots, n_y$.

**error** — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

# 4 Error Codes

## Fatal errors (error%level = 3):

| error%code | Description |
|---|---|
| **301** | An input argument has an invalid value. |
| **302** | An array argument has an invalid shape. |
| **303** | Array arguments have inconsistent shapes. |

### Failures (error%level = 2):

| error%code | Description |
|---|---|
| **201** | At least one evaluation point lies outside the region of definition of the interpolant. Values of q and its derivatives have been set to HUGE(1.0_*wp*) at all such points. |

# 5 Examples of Usage

Complete examples of the use of this procedure appear in Examples 1 and 4 of this module document.

# 6 Further Comments

## 6.1 Timing

The time taken to evaluate $q(x, y)$ at each evaluation point will depend in general on the distribution of the data points. If $x$ and $y$ are approximately uniformly distributed, then the time taken should be $O(1)$, i.e., independent of the number $m$ of data points used in the construction of the interpolant by nag_scat_2d_interp. At worst $O(m)$ time will be required.

# Procedure: nag_scat_3d_eval

## 1 Description

nag_scat_3d_eval computes values of a 3-d interpolant $q(x, y, z)$, and optionally its first partial derivatives, where $q(x, y, z)$ was generated by nag_scat_3d_interp.

This procedure is generic and may be used to evaluate the interpolant

- at a single point,
- at a set of $n$ scattered points,
- or on an $n_x \times n_y \times n_z$ grid of points.

## 2 Usage

```
USE nag_scat_interp
CALL nag_scat_3d_eval(interp, u, v, w, q  [, optional arguments])
```

### 2.1 Interfaces

Distinct interfaces are provided for the following cases:

Evaluation at a single point / at scattered points / at points on a grid

**Single point:** u, v, w, q and the optional arguments qx, qy and qz must all be scalars.

**Scattered points:** u, v, w, q and the optional arguments qx, qy and qz must all be rank-1 arrays.

**Points on a grid:** u, v and w must be rank-1 arrays. q and the optional arguments qx, qy and qz must be rank-3 arrays.

## 3 Arguments

**Note.** All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**($n$)' is used in the argument descriptions to specify that the array x must have exactly $n$ elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$n \geq 1$ — the total number of evaluation points (for scattered points)
$n_x \geq 1$ — the number of grid points in the $x$-direction (for points on a grid)
$n_y \geq 1$ — the number of grid points in the $y$-direction (for points on a grid)
$n_z \geq 1$ — the number of grid points in the $z$-direction (for points on a grid)

### 3.1 Mandatory Arguments

**interp** — type(nag_scat_comm_*wp*), intent(in)

*Input:* a structure containing details of the interpolant $q(x, y, z)$ to be evaluated.

*Constraints:* interp must be as output from a previous call to nag_scat_3d_interp or nag_scat_3d_set.

**u / u($n$) / u($n_x$)** — real(kind=*wp*), intent(in)

*Input:* the $x$-coordinate(s) of the evaluation point(s).

*Constraints:*

If q is scalar, u must be scalar;

if q is a rank-1 array, u must also be a rank-1 array, with SIZE(u) = SIZE(q);

if q is a rank-3 array, u must be a rank-1 array, with SIZE(u) = SIZE(q,1).

**v / v($n$) / v($n_y$)** — real(kind=*wp*), intent(in)

*Input:* the $y$-coordinate(s) of the evaluation point(s).

*Constraints:*

If q is scalar, v must be scalar;

if q is a rank-1 array, v must also be a rank-1 array, with SIZE(v) = SIZE(q);

if q is a rank-3 array, v must be a rank-1 array, with SIZE(v) = SIZE(q,2).

**w / w($n$) / w($n_z$)** — real(kind=*wp*), intent(in)

*Input:* the $z$ coordinate(s) of the evaluation point(s).

*Constraints:*

If q is scalar, w must be scalar;

if q is a rank-1 array, w must also be a rank-1 array, with SIZE(w) = SIZE(q);

if q is a rank-3 array, w must be a rank-1 array, with SIZE(w) = SIZE(q,3).

**q / q($n$) / q($n_x, n_y, n_z$)** — real(kind=*wp*), intent(out)

*Output:* the value(s) of the interpolant at the evaluation point(s).

If q is a scalar (single evaluation point), $\mathtt{q} = q(\mathtt{u}, \mathtt{v}, \mathtt{w})$;

if q is a rank-1 array (scattered evaluation points), $\mathtt{q} = q(\mathtt{u}(i), \mathtt{v}(i), \mathtt{w}(i))$, for $i = 1, 2, \ldots, n$;

if q is a rank-3 array (evaluation on a grid), $\mathtt{q} = q(\mathtt{u}(j), \mathtt{v}(k), \mathtt{w}(l))$, for $j = 1, 2, \ldots, n_x$, $k = 1, 2, \ldots, n_y$, $l = 1, 2, \ldots, n_z$.

## 3.2 Optional Arguments

**Note.** Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

**qx / qx($n$) / qx($n_x, n_y, n_z$)** — real(kind=*wp*), intent(out), optional

*Output:* the value(s) of the first partial derivative $q_x$ of the interpolant at the evaluation point(s).

If qx is a scalar (single evaluation point), $\mathtt{qx} = q_x(\mathtt{u}, \mathtt{v}, \mathtt{w})$;

if qx is a rank-1 array (scattered evaluation points), $\mathtt{qx} = q_x(\mathtt{u}(i), \mathtt{v}(i), \mathtt{w}(i))$, for $i = 1, 2, \ldots, n$;

if qx is a rank-3 array (evaluation on a grid), $\mathtt{qx} = q_x(\mathtt{u}(j), \mathtt{v}(k), \mathtt{w}(l))$, for $j = 1, 2, \ldots, n_x$, $k = 1, 2, \ldots, n_y$, $l = 1, 2, \ldots, n_z$.

**qy / qy($n$) / qy($n_x, n_y, n_z$)** — real(kind=*wp*), intent(out), optional

*Output:* the value(s) of the first partial derivative $q_y$ of the interpolant at the evaluation point(s).

If qy is a scalar (single evaluation point), $\mathtt{qy} = q_y(\mathtt{u}, \mathtt{v}, \mathtt{w})$;

if qy is a rank-1 array (scattered evaluation points), $\mathtt{qy} = q_y(\mathtt{u}(i), \mathtt{v}(i), \mathtt{w}(i))$, for $i = 1, 2, \ldots, n$;

if qy is a rank-3 array (evaluation on a grid), $\mathtt{qy} = q_y(\mathtt{u}(j), \mathtt{v}(k), \mathtt{w}(l))$, for $j = 1, 2, \ldots, n_x$, $k = 1, 2, \ldots, n_y$, $l = 1, 2, \ldots, n_z$.

**qz / qz**($n$) **/ qz**($n_x, n_y, n_z$) — real(kind=*wp*), intent(out), optional

*Output:* the value(s) of the first partial derivative $q_z$ of the interpolant at the evaluation point(s).

If qz is a scalar (single evaluation point), qz $= q_z($u, v, w$)$;

if qz is a rank-1 array (scattered evaluation points), qz $= q_z($u$(i),$ v$(i),$ w$(i))$, for $i = 1, 2, \ldots, n$;

if qz is a rank-3 array (evaluation on a grid), qz $= q_z($u$(j),$ v$(k),$ w$(l))$, for $j = 1, 2, \ldots, n_x$, $k = 1, 2, \ldots, n_y$, $l = 1, 2, \ldots, n_z$.

**error** — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

# 4 Error Codes

## Fatal errors (error%level = 3):

| error%code | Description |
|---|---|
| **301** | An input argument has an invalid value. |
| **302** | An array argument has an invalid shape. |
| **303** | Array arguments have inconsistent shapes. |

## Failures (error%level = 2):

| error%code | Description |
|---|---|
| **201** | At least one evaluation point lies outside the region of definition of the interpolant. Values of q and its derivatives have been set to HUGE(1.0_*wp*) at all such points. |

# 5 Examples of Usage

Complete examples of the use of this procedure appear in Examples 2 and 5 of this module document.

# 6 Further Comments

## 6.1 Timing

The time taken to evaluate $q(x, y, z)$ at each evaluation point will depend in general on the distribution of the data points. If $x$, $y$ and $z$ are approximately uniformly distributed, then the time taken should be $O(1)$, i.e., independent of the number $m$ of data points used in the construction of the interpolant by nag_scat_3d_interp. At worst $O(m)$ time will be required.

# Procedure: nag_scat_2d_set

## 1 Description

nag_scat_2d_set creates a structure of type nag_scat_comm_*wp* representing a 2-d interpolant $q(x, y)$, generated using a modified Shepard method (Renka [3]), of the set of scattered data points $(x_i, y_i, f_i)$, for $i = 1, 2, \ldots, m$.

The input arguments to this procedure will generally have been derived from a previous call to nag_scat_2d_interp followed by a call to nag_scat_extract.

This procedure and nag_scat_extract together provide a means of saving the details of an interpolant to file and then recovering them again. This usage is illustrated in Example 3 and Example 4 of this module document.

## 2 Usage

```
USE nag_scat_interp
CALL nag_scat_2d_set(x,y,f,iq,rq,interp  [, optional arguments])
```

## 3 Arguments

**Note.** All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**($n$)' is used in the argument descriptions to specify that the array **x** must have exactly $n$ elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

$m \geq 6$ — the number of data points

### 3.1 Mandatory Arguments

**x**($m$) — real(kind=*wp*), intent(in)

**y**($m$) — real(kind=*wp*), intent(in)

*Input:* the values $x_i$ and $y_i$, for $i = 1, 2, \ldots, m$.

**f**($m$) — real(kind=*wp*), intent(in)

*Input:* the values $f_i$ for $i = 1, 2, \ldots, m$.

**iq**($2m$) — integer, intent(in)

*Input:* integer data defining the interpolant $q(x, y)$.

**rq**($6m + 5$) — real(kind=*wp*), intent(in)

*Input:* real data defining the interpolant $q(x, y)$.

**interp** — type(nag_scat_comm_*wp*), intent(out)

*Output:* a structure containing details of the interpolant $q(x, y)$ stored. This structure may be passed to the procedure nag_scat_2d_eval to evaluate $q(x, y)$ at given points.

*Note:* to reduce the risk of corrupting the data accidentally, the components of this structure are private; details of $q(x, y)$ may be extracted by calling nag_scat_extract.

The procedure allocates roughly $9m + 5$ real(kind=*wp*) and $2m$ integer elements of storage to the structure. If you wish to deallocate this storage when the structure is no longer required, you must call the procedure nag_deallocate, as illustrated in Example 3 of this module document.

### 3.2 Optional Argument

**error** — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

## 4 Error Codes

### Fatal errors (error%level = 3):

| error%code | Description |
|---|---|
| **302** | An array argument has an invalid shape. |
| **303** | Array arguments have inconsistent shapes. |
| **320** | The procedure was unable to allocate enough memory. |

## 5 Examples of Usage

A complete example of the use of this procedure appears in Example 4 of this module document.

## 6 Further Comments

### 6.1 Algorithmic Detail

This procedure may be used for example to initialize an interpolant with data which has been read from file. The procedure nag_scat_extract may be used to extract the data to be written to file.

# Procedure: nag_scat_3d_set

## 1 Description

nag_scat_3d_set creates a structure of type nag_scat_comm_*wp* representing a 3-d interpolant $q(x, y, z)$, generated using a modified Shepard method (Renka [3]), of the set of scattered data points $(x_i, y_i, z_i, f_i)$, for $i = 1, 2, \ldots, m$.

The input arguments to this procedure will generally have been derived from a previous call to nag_scat_3d_interp followed by a call to nag_scat_extract.

This procedure and nag_scat_extract together provide a means of saving the details of an interpolant to file and then recovering them again. This usage is illustrated in Example 3 and Example 4 of this module document.

## 2 Usage

```
USE nag_scat_interp
CALL nag_scat_3d_set(x,y,z,f,iq,rq,interp  [, optional arguments])
```

## 3 Arguments

**Note.** All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly $n$ elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

$m \geq 10$ — the number of data points

### 3.1 Mandatory Arguments

**x**($m$) — real(kind=*wp*), intent(in)
**y**($m$) — real(kind=*wp*), intent(in)
**z**($m$) — real(kind=*wp*), intent(in)
*Input:* the values $x_i$, $y_i$ and $z_i$, for $i = 1, 2, \ldots, m$.

**f**($m$) — real(kind=*wp*), intent(in)
*Input:* the values $f_i$ for $i = 1, 2, \ldots, m$.

**iq**($2m$) — integer, intent(in)
*Input:* integer data defining the interpolant $q(x, y, z)$.

**rq**($10m + 7$) — real(kind=*wp*), intent(in)
*Input:* real data defining the interpolant $q(x, y, z)$.

**interp** — type(nag_scat_comm_*wp*), intent(out)
*Output:* a structure containing details of the interpolant $q(x, y, z)$ stored. This structure may be passed to the procedure nag_scat_3d_eval to evaluate $q(x, y, z)$ at given points.

*Note:* to reduce the risk of corrupting the data accidentally, the components of this structure are private; details of $q(x, y, z)$ may be extracted by calling nag_scat_extract.

The procedure allocates roughly $14m + 7$ real(kind=*wp*) and $2m$ integer elements of storage to the structure. If you wish to deallocate this storage when the structure is no longer required, you must call the procedure nag_deallocate, as illustrated in Example 3 of this module document.

### 3.2 Optional Argument

**error** — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

## 4 Error Codes

### Fatal errors (error%level = 3):

| error%code | Description |
|---|---|
| **302** | An array argument has an invalid shape. |
| **303** | Array arguments have inconsistent shapes. |
| **320** | The procedure was unable to allocate enough memory. |

## 5 Examples of Usage

A complete example of the use of this procedure appears in Example 5 of this module document.

## 6 Further Comments

### 6.1 Algorithmic Detail

This procedure may be used for example to initialize an interpolant with data which has been read from file. The procedure nag_scat_extract may be used to extract the data to be written to file.

# Procedure: nag_scat_extract

## 1 Description

nag_scat_extract extracts details of a 2-d or 3-d modified Shepard interpolant from a structure of type nag_scat_comm_*wp*.

Since the sizes of the array components of the structure may not be known prior to a call to this procedure the arguments which return these array components are pointers, which are allocated internally. It is your responsibility to deallocate this storage.

The procedures nag_scat_2d_set, nag_scat_3d_set and this procedure together provide a means of saving the details of a 2-d or 3-d interpolant to file and then recovering them again. This usage is illustrated in Examples 3 and 4 of this module document.

## 2 Usage

```
USE nag_scat_interp
CALL nag_scat_extract(interp  [, optional arguments])
```

## 3 Arguments

### 3.1 Mandatory Argument

**interp** — type(nag_scat_comm_*wp*), intent(in)

*Input:* a structure containing details of the interpolant.

*Constraints:* interp must be as output from a previous call to nag_scat_2d_interp, nag_scat_3d_interp, nag_scat_2d_set or nag_scat_3d_set.

### 3.2 Optional Arguments

**Note.** Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

**x**(:) — real(kind=*wp*), pointer, optional

*Output:* the $x$-coordinates of the interpolated data points.

*Note:* this array is allocated by this procedure. It should be deallocated when no longer required.

**y**(:) — real(kind=*wp*), pointer, optional

*Output:* the $y$-coordinates of the interpolated data points.

*Note:* this array is allocated by this procedure. It should be deallocated when no longer required.

**z**(:) — real(kind=*wp*), pointer, optional

*Output:* the $z$ coordinates of the interpolated data points.

*Constraints:* this argument must not be present if the structure interp was created by one of the 2-d procedures nag_scat_2d_interp or nag_scat_2d_set.

*Note:* this array is allocated by this procedure. It should be deallocated when no longer required.

**f**(:) — real(kind=*wp*), pointer, optional

*Output:* the values at the data points.

*Note:* this array is allocated by this procedure. It should be deallocated when no longer required.

**iq**(:) — integer, pointer, optional

*Output:* integer data defining the interpolant.

*Note:* this array is allocated by this procedure. It should be deallocated when no longer required.

**rq**(:) — real(kind=*wp*), pointer, optional

*Output:* real data defining the interpolant.

*Note:* this array is allocated by this procedure. It should be deallocated when no longer required.

**error** — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

# 4 Error Codes

### Fatal errors (error%level = 3):

| error%code | Description |
|---|---|
| **301** | An input argument has an invalid value. |
| **304** | Invalid presence of an optional argument. |
| **320** | The procedure was unable to allocate enough memory. |

# 5 Examples of Usage

A complete example of the use of this procedure appears in Example 3 of this module document.

# 6 Further Comments

## 6.1 Algorithmic Detail

This procedure may be used for example to extract data from the structure `interp` in order to write to file. One of the procedures `nag_scat_2d_set` or `nag_scat_3d_set` may then be used to initialize a new interpolant with this data.

If called with no optional arguments this procedure merely checks that the structure `interp` has been created by one of the interpolant generation procedures of this module.

# Derived Type: nag_scat_comm_*wp*

**Note.** The names of derived types containing real/complex components are precision dependent. For double precision the name of this type is **nag_scat_comm_dp**. For single precision the name is **nag_scat_comm_sp**. Please read the Users' Note for your implementation to check which precisions are available.

## 1 Description

The derived type nag_scat_comm_*wp* is used to represent a 2-d or 3-d interpolant generated using a modified Shepard method (Renka [3]).

The generation procedures nag_scat_2d_interp, nag_scat_3d_interp, nag_scat_2d_set and nag_scat_3d_set return structures of this type suitable for passing to the evaluation procedures nag_scat_2d_eval, nag_scat_3d_eval or nag_scat_extract.

These generation procedures allocate storage to the pointer components of the structure. For details of the amount of storage allocated see the description of the argument interp in the relevant procedure document.

If you wish to deallocate the storage when the structure is no longer required, you must call the generic deallocation procedure nag_deallocate, which is described in the module document nag_lib_support (1.1).

The generation procedures check whether the structure has already had storage allocated to it in a previous call; if it has, it deallocates that storage before allocating the storage required for the new call.

The components of this type are private.

## 2 Type Definition

```
type nag_scat_comm_wp
  private
  .
  .
  .
end type nag_scat_comm_wp
```

## 3 Components

In order to reduce the risk of accidental data corruption the components of this type are private and may not be accessed directly.

The procedures nag_scat_2d_set, nag_scat_3d_set and nag_scat_extract may be used to initialize and extract data from structures of the type.

# Example 1: Interpolation of scattered data in 2-d

Given a set of scattered data points in 2-d generate a modified Shepard interpolant $q(x, y)$ and evaluate it and its first partial derivatives at a sample of points on a rectangular grid.

## 1 Program Text

**Note.** The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_scat_interp_ex01

  ! Example Program Text for nag_scat_interp
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_scat_interp, ONLY : nag_scat_comm_wp => nag_scat_comm_dp, &
   nag_scat_2d_interp, nag_scat_2d_eval, nag_deallocate
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND, REAL
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j, m, nx, ny
  REAL (wp) :: dx, dy, xmax, xmin, ymax, ymin
  TYPE (nag_scat_comm_wp) :: interp
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: f(:), q(:,:), qx(:,:), qy(:,:), u(:), v(:), &
   x(:), y(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_scat_interp_ex01'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m
  READ (nag_std_in,*) nx, xmin, xmax
  READ (nag_std_in,*) ny, ymin, ymax

  ALLOCATE (x(m),y(m),f(m),u(nx),v(ny),q(nx,ny),qx(nx,ny), &
   qy(nx,ny))                  ! Allocate storage

  READ (nag_std_in,*) x
  READ (nag_std_in,*) y
  READ (nag_std_in,*) f

  ! Determine the modified Shepard interpolant.

  CALL nag_scat_2d_interp(x,y,f,interp)

  ! Evaluate the interpolant and its first partial derivatives at
  ! nx*ny points over the domain [xmin, xmax] x [ymin, ymax].

  dx = (xmax-xmin)/REAL(nx-1,kind=wp)
  dy = (ymax-ymin)/REAL(ny-1,kind=wp)
  u = (/ (xmin+(i-1)*dx,i=1,nx-1), xmax/)
  v = (/ (ymin+(j-1)*dy,j=1,ny-1), ymax/)

  CALL nag_scat_2d_eval(interp,u,v,q,qx=qx,qy=qy)
```

```
      WRITE (nag_std_out,'(/1X,A)') 'Values of interpolant q:'
      WRITE (nag_std_out,'(/11X,A,7F8.2)') 'x', u
      WRITE (nag_std_out,*) '     y'

      DO j = ny, 1, -1
        WRITE (nag_std_out,'(1X,F8.2,3X,7F8.2)') v(j), q(:,j)
      END DO

      WRITE (nag_std_out,'(/1X,A)') 'Values of derivative qx:'
      WRITE (nag_std_out,'(/11X,A,7F8.2)') 'x', u
      WRITE (nag_std_out,*) '     y'

      DO j = ny, 1, -1
        WRITE (nag_std_out,'(1X,F8.2,3X,7F8.2)') v(j), qx(:,j)
      END DO

      WRITE (nag_std_out,'(/1X,A)') 'Values of derivative qy:'
      WRITE (nag_std_out,'(/11X,A,7F8.2)') 'x', u
      WRITE (nag_std_out,*) '     y'

      DO j = ny, 1, -1
        WRITE (nag_std_out,'(1X,F8.2,3X,7F8.2)') v(j), qy(:,j)
      END DO

      DEALLOCATE (x,y,f,u,v,q,qx,qy) ! Deallocate storage

      CALL nag_deallocate(interp)  ! Free structure allocated by NAG fl90

    END PROGRAM nag_scat_interp_ex01
```

# 2 Program Data

```
Example Program Data for nag_scat_interp_ex01
 30                                 : m
  7  3.0  21.0                      : nx, xmin, xmax
  6  2.0  17.0                      : ny, ymin, ymax
 11.16  12.85  19.85  19.72  15.91
  0.00  20.87   3.45  14.26  17.43
 22.80   7.58  25.00   0.00   9.66
  5.22  17.25  25.00  12.13  22.23
 11.52  15.20   7.54  17.32   2.14
  0.51  22.69   5.47  21.67   3.31 : End of x
  1.24   3.06  10.72   1.39   7.74
 20.00  20.00  12.78  17.87   3.46
 12.39   1.98  11.87   0.00  20.00
 14.66  19.57   3.87  10.79   6.21
  8.53   0.00  10.69  13.78  15.03
  8.37  19.63  17.13  14.36   0.33 : End of y
 22.15  22.11   7.97  16.83  15.30
 34.60   5.74  41.24  10.74  18.60
  5.47  29.87   4.40  58.20   4.73
 40.36   6.43   8.74  13.71  10.25
 15.74  21.60  19.31  12.11  53.10
 49.43   3.25  28.63   5.52  44.08 : End of f
```

# 3 Program Results

```
Example Program Results for nag_scat_interp_ex01

Values of interpolant q:

          x    3.00    6.00    9.00   12.00   15.00   18.00   21.00
     y
   17.00      40.79   27.68   21.12   14.90   12.14    9.64    5.87
   14.00      47.14   37.08   24.47   16.70   13.69   11.29    6.27
   11.00      39.74   25.55   16.78   13.78   13.11   10.59    6.99
    8.00      36.16   20.70   17.08   15.76   15.51   12.90    9.61
    5.00      37.06   27.63   22.72   20.76   18.93   16.74   12.70
    2.00      43.45   33.70   26.20   22.25   21.01   18.76   15.15

Values of derivative qx:

          x    3.00    6.00    9.00   12.00   15.00   18.00   21.00
     y
   17.00      -5.33   -2.60   -2.35   -1.63   -0.55   -1.08   -1.21
   14.00      -6.35   -4.09   -3.52   -1.60   -0.74   -1.34   -1.08
   11.00      -5.70   -4.43   -1.02   -1.02    0.00   -1.60   -0.57
    8.00      -7.05   -2.71    0.03   -0.71   -0.25   -1.37   -0.75
    5.00      -4.29   -1.97   -1.17   -0.50   -0.53   -1.07   -1.35
    2.00      -3.62   -2.36   -2.27   -0.37   -0.47   -1.07   -1.26

Values of derivative qy:

          x    3.00    6.00    9.00   12.00   15.00   18.00   21.00
     y
   17.00      -5.11   -5.00   -4.07   -2.48   -1.58   -0.76   -0.15
   14.00       2.44    1.99    2.43    1.49    0.05   -0.37   -0.40
   11.00       2.74    2.59    0.91   -0.24   -0.16   -0.01   -0.25
    8.00       0.36    0.24   -0.77   -0.87   -0.96   -1.15   -1.28
    5.00      -1.92   -3.65   -2.51   -1.89   -1.26   -1.17   -1.06
    2.00      -1.56   -0.32    0.13    0.83   -0.28   -0.37   -0.33
```

# Example 2: Interpolation of scattered data in 3-d

Given a set of scattered data points in 3-d generate a modified Shepard interpolant $q(x, y, z)$ and evaluate it and its first partial derivatives at the mean point in the data set.

## 1 Program Text

**Note.** The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_scat_interp_ex02

  ! Example Program Text for nag_scat_interp
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_scat_interp, ONLY : nag_scat_comm_wp => nag_scat_comm_dp, &
   nag_scat_3d_interp, nag_scat_3d_eval, nag_deallocate
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND, REAL, SUM
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: m
  REAL (wp) :: q, qx, qy, qz, u, v, w
  TYPE (nag_scat_comm_wp) :: interp
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: f(:), x(:), y(:), z(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_scat_interp_ex02'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m

  ALLOCATE (x(m),y(m),z(m),f(m)) ! Allocate storage

  READ (nag_std_in,*) x
  READ (nag_std_in,*) y
  READ (nag_std_in,*) z
  READ (nag_std_in,*) f

  ! Determine the modified Shepard interpolant.

  CALL nag_scat_3d_interp(x,y,z,f,interp)

  ! Evaluate the interpolant and its first partial derivatives
  ! at the mean point in the data set.

  u = SUM(x)/REAL(m,kind=wp)
  v = SUM(y)/REAL(m,kind=wp)
  w = SUM(z)/REAL(m,kind=wp)

  CALL nag_scat_3d_eval(interp,u,v,w,q,qx=qx,qy=qy,qz=qz)

  WRITE (nag_std_out,'(/1X,A,F10.4)') 'u  =', u
  WRITE (nag_std_out,'(1X,A,F10.4)') 'v  =', v
  WRITE (nag_std_out,'(1X,A,F10.4)') 'w  =', w
  WRITE (nag_std_out,'(1X,A,F10.4)') 'q  =', q
```

```
      WRITE (nag_std_out,'(1X,A,F10.4)') 'qx =', qx
      WRITE (nag_std_out,'(1X,A,F10.4)') 'qy =', qy
      WRITE (nag_std_out,'(1X,A,F10.4)') 'qz =', qz

      DEALLOCATE (x,y,z,f)         ! Deallocate storage

      CALL nag_deallocate(interp)  ! Free structure allocated by NAG fl90

    END PROGRAM nag_scat_interp_ex02
```

# 2 Program Data

```
Example Program Data for nag_scat_interp_ex02
 30                              : m
 0.80  0.23  0.18  0.58  0.64
 0.88  0.30  0.87  0.04  0.62
 0.87  0.62  0.86  0.87  0.49
 0.12  0.02  0.62  0.49  0.36
 0.62  0.01  0.41  0.17  0.51
 0.85  0.20  0.04  0.31  0.88 : End of x
 0.23  0.88  0.43  0.95  0.69
 0.35  0.10  0.09  0.02  0.90
 0.96  0.64  0.13  0.60  0.43
 0.61  0.71  0.93  0.54  0.56
 0.42  0.72  0.36  0.99  0.29
 0.05  0.20  0.67  0.63  0.27 : End of y
 0.37  0.05  0.04  0.62  0.20
 0.49  0.78  0.05  0.40  0.43
 0.24  0.45  0.47  0.46  0.13
 0.00  0.82  0.44  0.04  0.39
 0.97  0.45  0.52  0.65  0.59
 0.04  0.87  0.04  0.18  0.07 : End of z
 0.51  1.80  0.11  2.65  0.93
 0.72 -0.11  0.67  0.00  2.20
 3.17  0.74  0.64  1.07  0.22
 0.41  0.58  2.48  0.37  0.35
-0.20  0.78  0.11  2.82  0.14
 0.61 -0.25  0.59  0.50  0.71 : End of f
```

# 3 Program Results

```
 Example Program Results for nag_scat_interp_ex02

 u  =    0.4820
 v  =    0.5117
 w  =    0.3750
 q  =    0.3097
 qx =    0.5966
 qy =    1.5456
 qz =    0.0134
```

# Example 3: Extracting details of a 2-d modified Shepard interpolant

Generate a modified Shepard interpolant $q(x, y)$ using `nag_scat_2d_interp`. Extract and output details of $q(x, y)$ from the structure `interp` using `nag_scat_extract`.

## 1 Program Text

**Note.** The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_scat_interp_ex03

  ! Example Program Text for nag_scat_interp
  ! NAG f190, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_scat_interp, ONLY : nag_scat_comm_wp => nag_scat_comm_dp, &
   nag_scat_2d_interp, nag_scat_extract, nag_deallocate
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: m
  TYPE (nag_scat_comm_wp) :: interp
  ! .. Local Arrays ..
  INTEGER, POINTER :: iq(:)
  REAL (wp), ALLOCATABLE :: f(:), x(:), y(:)
  REAL (wp), POINTER :: rq(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_scat_interp_ex03'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m

  ALLOCATE (x(m),y(m),f(m))    ! Allocate storage

  READ (nag_std_in,*) x
  READ (nag_std_in,*) y
  READ (nag_std_in,*) f

  ! Determine the modified Shepard interpolant.

  CALL nag_scat_2d_interp(x,y,f,interp)

  ! Extract and output the components iq and rq from the structure
  ! interp. These contain integer and real data defining the
  ! interpolant q.

  CALL nag_scat_extract(interp,iq=iq,rq=rq)

  WRITE (nag_std_out,'(/1X,A)') 'The array iq:'
  WRITE (nag_std_out,'(5I10)') iq
  WRITE (nag_std_out,'(/1X,A)') 'The array rq:'
  WRITE (nag_std_out,'(5ES15.4)') rq

  DEALLOCATE (x,y,f,iq,rq)     ! Deallocate storage
```

```
      CALL nag_deallocate(interp)  ! Free structure allocated by NAG fl90

    END PROGRAM nag_scat_interp_ex03
```

# 2 Program Data

```
Example Program Data for nag_scat_interp_ex03
 6                                  : m
 7.9512E-01  2.2572E-01  3.7128E-01
 2.2504E-01  8.7874E-01  4.7473E-02 : x
 1.8057E-01  4.3277E-01  3.9370E-02
 5.7517E-01  9.5461E-01  6.2410E-01 : y
 6.3147E-01 -5.7411E-02  8.6245E-02
-2.3087E-01 -3.6669E-01 -3.5747E-01 : f
```

# 3 Program Results

```
 Example Program Results for nag_scat_interp_ex03

 The array iq:
         1         0         0         0         0
         0         2         3         4         5
         6         6

 The array rq:
     4.7473E-02     3.9370E-02     8.3127E-01     9.1524E-01     1.0976E+00
     8.3126E-01     7.6863E-01     1.2047E+00     6.2843E-01     1.2047E+00
     8.8027E-01     1.0001E+00     1.0631E-04    -2.0001E+00     1.5903E+00
     7.7771E-02     1.0001E+00     1.0631E-04    -2.0001E+00     4.5138E-01
    -9.3114E-01     1.0001E+00     1.0631E-04    -2.0001E+00     7.4250E-01
     6.4256E-01     1.0001E+00     1.0631E-04    -2.0001E+00     4.5004E-01
    -1.5008E+00     1.0001E+00     1.0631E-04    -2.0001E+00     1.7577E+00
    -3.0185E+00     1.0001E+00     1.0631E-04    -2.0001E+00     9.4861E-02
    -1.6965E+00
```

# Example 4: Initialization of a 2-d modified Shepard interpolant

Initialize a structure `interp` with details of a 2-d modified Shepard interpolant which are read from file. Evaluate the interpolant at the mean data point.

## 1 Program Text

**Note.** The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_scat_interp_ex04

  ! Example Program Text for nag_scat_interp
  ! NAG f190, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_scat_interp, ONLY : nag_scat_comm_wp => nag_scat_comm_dp, &
   nag_scat_2d_set, nag_scat_2d_eval, nag_deallocate
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND, REAL, SUM
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: m
  REAL (wp) :: q, qx, qy, u, v
  TYPE (nag_scat_comm_wp) :: interp
  ! .. Local Arrays ..
  INTEGER, ALLOCATABLE :: iq(:)
  REAL (wp), ALLOCATABLE :: f(:), rq(:), x(:), y(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_scat_interp_ex04'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m

  ALLOCATE (x(m),y(m),f(m),iq(2*m),rq(6*m+5)) ! Allocate storage

  READ (nag_std_in,*) x
  READ (nag_std_in,*) y
  READ (nag_std_in,*) f
  READ (nag_std_in,*) iq
  READ (nag_std_in,*) rq

  ! Initialize the structure interp.

  CALL nag_scat_2d_set(x,y,f,iq,rq,interp)

  ! Evaluate the interpolant and its first partial derivatives
  ! at the mean point in the data set.

  u = SUM(x)/REAL(m,kind=wp)
  v = SUM(y)/REAL(m,kind=wp)

  CALL nag_scat_2d_eval(interp,u,v,q,qx=qx,qy=qy)

  WRITE (nag_std_out,'(/1X,A,F10.4)') 'u  =', u
  WRITE (nag_std_out,'(1X,A,F10.4)') 'v  =', v
```

```
      WRITE (nag_std_out,'(1X,A,F10.4)') 'q  =', q
      WRITE (nag_std_out,'(1X,A,F10.4)') 'qx =', qx
      WRITE (nag_std_out,'(1X,A,F10.4)') 'qy =', qy

      DEALLOCATE (x,y,f,iq,rq)     ! Deallocate storage

      CALL nag_deallocate(interp)  ! Free structure allocated by NAG fl90

    END PROGRAM nag_scat_interp_ex04
```

# 2 Program Data

```
Example Program Data for nag_scat_interp_ex04
 6                                  : m
 7.9512E-01  2.2572E-01  3.7128E-01
 2.2504E-01  8.7874E-01  4.7473E-02 : x
 1.8057E-01  4.3277E-01  3.9370E-02
 5.7517E-01  9.5461E-01  6.2410E-01 : y
 5.0194E-01 -9.6860E-02 -4.2321E-04
-2.7011E-01 -4.6032E-01 -3.5962E-01 : f
 1  0  0  0  0  0
 2  3  4  5  6  6                   : iq
 4.7473E-02  3.9370E-02  8.3127E-01
 9.1524E-01  1.0976E+00  8.3127E-01
 7.6865E-01  1.2047E+00  6.2844E-01
 1.2047E+00  8.8027E-01  1.1445E+00
 9.2579E-02 -2.0713E+00  1.5270E+00
 1.7703E-01  1.1445E+00  9.2579E-02
-2.0713E+00  2.4701E-01 -9.2040E-01
 1.1445E+00  9.2579E-02 -2.0713E+00
 5.4377E-01  7.2272E-01  1.1445E+00
 9.2579E-02 -2.0713E+00  2.5863E-01
-1.5104E+00  1.1445E+00  9.2579E-02
-2.0713E+00  1.7901E+00 -3.0217E+00
 1.1445E+00  9.2579E-02 -2.0713E+00
-1.4327E-01 -1.7295E+00             : rq
```

# 3 Program Results

```
 Example Program Results for nag_scat_interp_ex04

 u  =    0.4239
 v  =    0.4678
 q  =   -0.0371
 qx =    0.7039
 qy =   -1.0470
```

# Example 5: Initialization of a 3-d modified Shepard interpolant

Initialize a structure `interp` with details of a 3-d modified Shepard interpolant which are read from file. Evaluate the interpolant at the mean data point.

## 1 Program Text

**Note.** The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_scat_interp_ex05

  ! Example Program Text for nag_scat_interp
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_scat_interp, ONLY : nag_scat_comm_wp => nag_scat_comm_dp, &
   nag_scat_3d_set, nag_scat_3d_eval, nag_deallocate
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND, REAL, SUM
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: m
  REAL (wp) :: q, qx, qy, qz, u, v, w
  TYPE (nag_scat_comm_wp) :: interp
  ! .. Local Arrays ..
  INTEGER, ALLOCATABLE :: iq(:)
  REAL (wp), ALLOCATABLE :: f(:), rq(:), x(:), y(:), z(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_scat_interp_ex05'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) m

  ALLOCATE (x(m),y(m),z(m),f(m),iq(2*m),rq(10*m+7)) ! Allocate storage

  READ (nag_std_in,*) x
  READ (nag_std_in,*) y
  READ (nag_std_in,*) z
  READ (nag_std_in,*) f
  READ (nag_std_in,*) iq
  READ (nag_std_in,*) rq

  ! Initialize the structure interp.

  CALL nag_scat_3d_set(x,y,z,f,iq,rq,interp)

  ! Evaluate the interpolant and its first partial derivatives
  ! at the mean point in the data set.

  u = SUM(x)/REAL(m,kind=wp)
  v = SUM(y)/REAL(m,kind=wp)
  w = SUM(z)/REAL(m,kind=wp)

  CALL nag_scat_3d_eval(interp,u,v,w,q,qx=qx,qy=qy,qz=qz)
```

```
      WRITE (nag_std_out,'(/1X,A,F10.4)') 'u  =', u
      WRITE (nag_std_out,'(1X,A,F10.4)') 'v  =', v
      WRITE (nag_std_out,'(1X,A,F10.4)') 'w  =', w
      WRITE (nag_std_out,'(1X,A,F10.4)') 'q  =', q
      WRITE (nag_std_out,'(1X,A,F10.4)') 'qx =', qx
      WRITE (nag_std_out,'(1X,A,F10.4)') 'qy =', qy
      WRITE (nag_std_out,'(1X,A,F10.4)') 'qz =', qz

      DEALLOCATE (x,y,z,f,iq,rq)   ! Deallocate storage

      CALL nag_deallocate(interp)  ! Free structure allocated by NAG fl90

    END PROGRAM nag_scat_interp_ex05
```

# 2 Program Data

```
Example Program Data for nag_scat_interp_ex05
 10                                                          : m
 7.9512E-01  2.2572E-01  3.7128E-01  2.2504E-01  8.7874E-01
 4.7473E-02  1.8057E-01  4.3277E-01  3.9370E-02  5.7517E-01 : x
 9.5461E-01  6.2410E-01  6.3907E-01  6.8812E-01  2.0124E-01
 8.8101E-01  3.4762E-01  4.9382E-01  3.0297E-01  1.0393E-01 : y
 7.8215E-01  8.7325E-01  8.8944E-02  5.3814E-02  3.8521E-02
 2.2838E-02  4.0199E-01  6.2099E-01  8.9889E-01  4.2641E-01 : z
 8.5303E-01 -4.8210E-01  2.0275E-01  1.1794E-01  7.1414E-01
 3.6954E-02  1.5945E-03  3.7873E-02 -6.4920E-01  1.6343E-01 : f
 1  0  0  0  0  0  0  0  0  0
 2  3  4  5  6  7  8  9 10 10                               : iq
 3.9370E-02  1.0393E-01  2.2838E-02  8.3937E-01  8.5068E-01
 8.7606E-01  1.2652E+00  1.2551E+00  1.4322E+00  9.6706E-01
 1.0191E+00  1.6007E+00  1.2687E+00  9.7971E-01  7.2177E-01
 1.6007E+00  1.1497E+00  2.0075E-03  3.2175E-03  1.5390E-03
 2.2483E-03  2.3191E-04 -2.0074E-03  1.9354E+00  4.5128E-01
-5.8743E-01  1.6834E+00  9.6118E-01 -3.8724E-01  9.1498E-01
 2.2237E-01 -9.4365E-01  9.8306E-01  4.4224E-02 -1.5185E+00
 1.6834E+00  9.6118E-01 -3.8724E-01  9.1498E-01  2.2237E-01
-9.4365E-01  7.6991E-01 -1.8606E-03  9.8261E-02  1.6834E+00
 9.6118E-01 -3.8724E-01  9.1498E-01  2.2237E-01 -9.4365E-01
 2.9252E-01 -1.8823E-01  4.1657E-02  1.6834E+00  9.6118E-01
-3.8724E-01  9.1498E-01  2.2237E-01 -9.4365E-01  2.0115E+00
 8.1378E-01  5.6039E-01  1.6834E+00  9.6118E-01 -3.8724E-01
 9.1498E-01  2.2237E-01 -9.4365E-01 -1.4825E-01 -5.1517E-01
-1.9456E-02  1.6834E+00  9.6118E-01 -3.8724E-01  9.1498E-01
 2.2237E-01 -9.4365E-01  1.3411E-01  1.1017E-01 -7.3186E-01
 1.6834E+00  9.6118E-01 -3.8724E-01  9.1498E-01  2.2237E-01
-9.4365E-01  1.3241E+00  2.8804E-01 -8.8192E-01  1.6834E+00
 9.6118E-01 -3.8724E-01  9.1498E-01  2.2237E-01 -9.4365E-01
 7.0443E-02  1.1952E-01 -1.8088E+00  1.6834E+00  9.6118E-01
-3.8724E-01  9.1498E-01  2.2237E-01 -9.4365E-01  1.2508E+00
 6.8361E-01 -4.7108E-01                                     : rq
```

# 3 Program Results

```
Example Program Results for nag_scat_interp_ex05

u  =    0.3771
v  =    0.5236
w  =    0.4208
q  =    0.1231
qx =    0.9882
qy =    0.1656
qz =   -0.5487
```

# References

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

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

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

[4] Renka R J (1988) Algorithm 660: QSHEP2D: Quadratic Shepard method for bivariate interpolation of scattered data *ACM Trans. Math. Software* **14** 149–150

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

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