# Module 8.3: nag_spline_2d
# Two-dimensional Spline Fitting

nag_spline_2d provides procedures for computing and evaluating spline approximations to arbitrary data sets in two dimensions.

## Contents

# Introduction

This module is concerned with a two-dimensional bicubic spline $s(x, y)$, which is defined for $(x, y) \in [a, b] \times [c, d]$, and expressed in its B-spline series representation

$$s(x, y) = \sum_{i=1}^{p-4} \sum_{j=1}^{q-4} \kappa_{ij} M_i(x) N_j(y),$$

where $M_i(x)$ and $N_j(y)$ denote normalized cubic B-splines (see Hayes [11]), the former defined on the knots $\lambda_i, \lambda_{i+1}, \ldots, \lambda_{i+4}$, and the latter on the knots $\mu_j, \mu_{j+1}, \ldots, \mu_{j+4}$. The module defines the derived type `nag_spline_2d_comm_wp` to represent $s(x, y)$ in the above form.

Such a spline may be used to interpolate (pass exactly through) a given set of data points $(x_i, y_i, f_i)$, for $i = 1, 2, \ldots, m$. The procedure `nag_spline_2d_interp` generates such an interpolant for data points lying on a rectangular mesh. Alternatively, a spline may be used to approximate the points, without actually passing through them. In the latter case it is useful to have some measure of the accuracy of the fit. For this purpose we define the sum of squares of the weighted residuals

$$\theta = \sum_{i=1}^{m} w_i^2 (f_i - s(x_i, y_i))^2,$$

where the weights $w_i$, $i = 1, 2, \ldots, m$ may be used to ensure that $f$-values known to be more accurate than others have a greater influence on $\theta$.

A *least-squares* spline approximation is one for which the coefficients $\kappa_{ij}$ have been chosen in order to minimise $\theta$. Typically, a least-squares spline approximation involves significantly fewer coefficients than the corresponding interpolating spline. Its use is much less liable to produce unwanted fluctuations, and so can often provide a better approximation to the function underlying the data. The procedure `nag_spline_2d_lsq_fit` computes a weighted least-squares fit with given interior knots.

A much more automatic fitting procedure can be derived by choosing both the interior knots and the coefficients $\kappa_{ij}$ in order to optimise some measure of the smoothness of $s(x, y)$, subject to $\theta$ being less than a given threshold. An algorithm of this type is implemented by `nag_spline_2d_auto_fit` for scattered or regular data.

The spline is assumed to have a total of $p$ knots $\lambda_1, \lambda_2, \ldots, \lambda_p$ in the $x$-direction, and $q$ knots $\mu_1, \mu_2, \ldots, \mu_q$ in the $y$-direction. Of these, the first four and the last four in each direction are defined by

$$\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = a, \quad \lambda_{p-3} = \lambda_{p-2} = \lambda_{p-1} = \lambda_p = b,$$

$$\mu_1 = \mu_2 = \mu_3 = \mu_4 = c, \quad \mu_{q-3} = \mu_{q-2} = \mu_{q-1} = \mu_q = d.$$

The remaining *interior* knots $\lambda_5, \lambda_6, \ldots, \lambda_{p-4}$ and $\mu_5, \mu_6, \ldots, \mu_{q-4}$ are either automatically selected or specified through input arguments, depending on the spline generation procedure used. A knot in the $x$-direction is a value of $x$ at which the spline is allowed to be discontinuous in its third derivative with respect to $x$, though continuous up to its second derivative. This degree of continuity can be reduced, if required, by the use of coincident knots, provided that no more than four knots are chosen to coincide at any value of $x$. Two, or three, coincident knots allow loss of continuity in, respectively, the second and first derivatives. Four coincident knots split the spline surface into independent parts. Knots in the $y$-direction are similarly defined and constrained.

In addition to the derived type and procedures mentioned above, the module also provides procedures for the evaluation of the spline at scattered points or on a regular grid, and for computing a definite integral of the spline.

# Procedure: nag_spline_2d_auto_fit

## 1 Description

This procedure determines a smooth bicubic spline approximation $s(x, y)$ to a set of data points in two dimensions. It is a generic procedure and may be used for either of the following two classes of problem.

- Approximate the *scattered data points* $(x_i, y_i, f_i)$, with weights $w_i$, for $i = 1, 2, \ldots, m$. The weights are by default set to one, but may be set to other values by supplying the optional argument `wt`.
- Approximate the data points $(x_j, y_k, f_{jk})$, with weights $w_{jk} = 1$, for $j = 1, 2, \ldots, m_x$, and $k = 1, 2, \ldots, m_y$, which *lie on a grid*. The grid points must satisfy $x(1) < x(2) \cdots < x(m_x)$ and $y(1) < y(2) \cdots < y(m_y)$. Non-unit weights are not permitted.

The resulting spline $s(x, y)$ is defined in the region $[a, b] \times [c, d]$, where $a = \min_i x_i$, $b = \max_i x_i$, $c = \min_i y_i$ and $d = \max_i y_i$.

The total number of knots in each coordinate direction, and their values are chosen automatically by the procedure. The balance between closeness of fit and smoothness of the approximation $s(x, y)$ is controlled by means of the *smoothing factor* $S$. If $S$ is too large, the spline will be too smooth and information will be lost (underfit); for very large $S$ the procedure returns the weighted least-squares bicubic polynomial. If $S$ is too small, the spline will pick up too much noise (overfit); as $S$ tends to zero the approximation generated tends to an interpolating spline. Experimenting with values between these two extremes should result in a good compromise. (See Section 6.4 for advice.) Note, however, that this procedure, unlike `nag_spline_1d_auto_fit`, does not allow $S$ to be set *exactly* to zero.

## 2 Usage

```
USE nag_spline_2d
CALL nag_spline_2d_auto_fit(start, x, y, f, smooth, spline  [, optional arguments])
```

### 2.1 Interfaces

Distinct interfaces are provided for the following cases.

Scattered data points / Data points on a grid

**Scattered points:** `x`, `y` and `f` must all be rank-1 arrays.

**Points on a grid:** `x` and `y` must be rank-1 arrays, and `f` a rank-2 array.

## 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.

$m \geq 16$ — the total number of data points (for scattered points)

$m_x \geq 4$ — the number of grid points in the $x$-direction (for points on a grid)

$m_y \geq 4$ — the number of grid points in the $y$-direction (for points on a grid)

## 3.1 Mandatory Arguments

**start** — character(len=1), intent(in)

*Input:* specifies whether a *cold* or *warm* start is required.

If `start` = `'C'` or `'c'`, the procedure will build up the knot set starting from no interior knots. For this *cold* start no initialization of the argument `spline` is required.

If `start` = `'W'` or `'w'`, the procedure will restart the knot-placing strategy using the knots found in a previous call to the procedure. For this *warm* start the structure `spline` must be unchanged from that previous call.

*Note:* a warm start can save much time in searching for a satisfactory value of `smooth`.

*Constraints:* `start` = `'C'` or `'c'` on the first call to the procedure; `start` = `'C'`, `'c'`, `'W'`, or `'w'` on subsequent calls.

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

*Input:* the $x$-coordinates of the data points.

*Constraints:*

If `f` is a rank-1 array, `SIZE(x)` must equal `SIZE(f)`, and the elements of `x` must not all be equal;

if `f` is a rank-2 array, `SIZE(x)` must equal `SIZE(f,1)`, and the elements of `x` must be strictly increasing.

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

*Input:* the $y$-coordinates of the data points.

*Constraints:*

If `f` is a rank-1 array, `SIZE(y)` must equal `SIZE(f)`, and the elements of `y` must not all be equal;

if `f` is a rank-2 array, `SIZE(y)` must equal `SIZE(f,2)`, and the elements of `y` must be strictly increasing.

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

*Input:* the $f$-values at the data points.

If `f` is a rank-1 array (scattered data points), `f(`$i$`)` is the $f$-value corresponding to the data point $(\mathtt{x}(i), \mathtt{y}(i))$, for $i = 1, 2, \ldots, m$;

if `f` is a rank-2 array (data points on a grid), `f(`$j, k$`)` is the $f$-value corresponding to the data point $(\mathtt{x}(j), \mathtt{y}(k))$, for $j = 1, 2, \ldots, m_x$ and $k = 1, 2, \ldots, m_y$.

**smooth** — real(kind=*wp*), intent(in)

*Input:* the smoothing factor $S$.

*Note:* for advice on the choice of `smooth` see Section 6.4.

*Constraints:* `smooth` $> 0.0$.

**spline** — type(nag_spline_2d_comm_*wp*), intent(inout)

*Input:* a structure representing the spline.

If `start` = `'C'` or `'c'`, no initialization of `spline` is required.

If `start` = `'W'` or `'w'`, the structure must be as output from a previous call to this procedure.

*Output:* a structure containing details of the spline $s(x, y)$ generated. This structure may be passed to the procedure nag_spline_2d_eval to evaluate $s(x, y)$ at given points, or to nag_spline_2d_intg to compute its definite integral.

*Note:* to reduce the risk of corrupting the data accidentally, the components of this structure are private; details of the spline may be extracted by calling nag_spline_2d_extract.

The procedure allocates a maximum of roughly $11m(\sqrt{m} + 10)$ real(kind=*wp*) elements of storage to the structure when fitting to data at $m$ scattered points, and roughly $2(m_x + 12)(m_y + 10)$ elements when fitting to data on an $m_x \times m_y$ grid. 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.

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

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

*Note:* zero weights are permitted and the corresponding points will be ignored, except when determining the region of definition of the spline. Section 3.3 of the Chapter Introduction gives advice on the choice of weights.

*Default:* wt = 1.0.

*Constraints:* this argument may only be supplied to the interface for scattered data points (f of rank-1).

wt($i$) $\geq 0.0$, for $i = 1, 2, \ldots, m$.

The number of strictly positive weights must be at least 16.

**p** — integer, intent(out), optional

*Output:* the total number of knots $p$ chosen by this procedure in the $x$-direction.

**q** — integer, intent(out), optional

*Output:* the total number of knots $q$ chosen by this procedure in the $y$-direction.

**theta** — real(kind=*wp*), intent(out), optional

*Output:* the sum of squares of weighted residuals $\theta$, as described in the Module Introduction.

*Note:* if theta = 0.0, this is an interpolating spline. theta should equal smooth within a relative tolerance of 0.001 unless $p = q = 8$, when the spline has no interior knots and so is simply a bicubic polynomial. For knots to be inserted, smooth must be set to a value below the value of theta produced in this case.

**rank** — integer, intent(out), optional

*Output:* the rank of the system of equations used to compute the final spline (as determined by a suitable machine-dependent threshold).

*Note:* when rank = $(p-4)(q-4)$ the solution is unique; otherwise the system is rank-deficient and the minimum norm solution is computed. The latter case may be caused by too small a value of $S$.

*Constraints:* this argument may only be supplied to the interface for scattered data points (f of rank-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. |

## Failures (error%level = 2):

| error%code | Description |
|---|---|
| **201** | An iterative process has failed to converge.<br>The iterative process used to compute the coefficients of the approximating spline has failed to converge. This error exit may occur if `smooth` has been set very small. If the error persists with increased `smooth` consult NAG. |
| **202** | The termination criteria could not be met.<br>No more knots can be added because the number of B-spline coefficients $(p-4)(q-4)$ already exceeds the number of data points $m$. This error exit may occur if either of $S$ or $m$ is too small. |
| **203** | The termination criteria could not be met.<br>No more knots can be added because the additional knot would be indistinguishable from an old one. This error exit may occur if too large a weight has been given to an inaccurate data point, or if `smooth` is too small. |

If `error%level` = 2 a spline approximation is computed, but fails to satisfy the fitting criterion (see Section 6.1) — perhaps by only a small amount, however. If you wish to use this approximation you must supply the optional argument `error` with `error%halt_level` set to 3.

# 5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

# 6 Further Comments

## 6.1 Algorithmic Detail

The B-spline coefficients $\kappa_{ij}$ are determined as the solution of the following constrained minimisation problem:

$$\text{minimize} \quad \eta, \quad \text{subject to} \quad \theta \leq S,$$

where $\theta$, the sum of squares of weighted residuals, is defined as

$$\theta = \begin{cases} \displaystyle\sum_{i=1}^{m} w_i^2 (f_i - s(x_i, y_i))^2 & \text{Scattered data points (f of rank-1)} \\ \displaystyle\sum_{j=1}^{m_x} \sum_{k=1}^{m_y} (f_{jk} - s(x_j, y_k))^2 & \text{Data points on a grid (f of rank-2)} \end{cases}$$

and $\eta$ is a measure of the lack of smoothness of $s(x, y)$. The value of $\eta$ depends on the discontinuity jumps in $s(x, y)$ across lines joining the knots. It is zero only when there are no discontinuities and is positive otherwise, increasing with the size of the jumps (see Dierckx [6] for details).

By means of the parameter $S > 0$, the *smoothing factor*, you may control the balance between smoothness and closeness of fit, as measured by $\theta$.

Suitable knot sets are built up in stages (starting with no interior knots in the case of a cold start but with the knot sets found in a previous call if a warm start is chosen). At each stage, a bicubic spline is fitted to the data by least-squares and $\theta$ is computed. If $\theta > S$, a new knot is added to one knot set or the other so as to reduce $\theta$ at the next stage. The new knot is located in an interval where the fit is particularly poor. At some point in the computation the condition $\theta \leq S$ will be satisfied, and at that point the knot sets are accepted. The procedure then goes on to compute a spline which has these knot sets and which satisfies the full fitting criterion specified above. This spline is unique for data points lying on a rectangular grid. The theoretical solution has $\theta = S$. The spline is computed by an iterative scheme which is ended when $\theta = S$ within a relative tolerance of 0.001. The main part of each iteration consists of a linear least-squares computation of special form, done in a similarly stable and efficient manner as in `nag_spline_2d_lsq_fit`.

An exception occurs when the procedure finds at the start that, even with no interior knots ($p = 8$, $q = 8$), the least-squares spline already has a value of $\theta \leq S$. In this case, since this spline (which is simply a bicubic polynomial) also has an optimal value for the smoothness measure $\eta$, namely zero, it is returned at once as the trivial solution. It will usually mean that $S$ has been chosen too large.

For further details of the algorithm and its use, see Dierckx [6], Dierckx [7].

## 6.2 Accuracy

On successful exit, the approximation returned is such that its residual norm is equal to the smoothing factor $S$, up to a relative tolerance of 0.001, except that if $p = 8$ and $q = 8$, it may be significantly less than $S$: in this case the computed spline is simply a weighted least-squares bicubic polynomial approximation, i.e., a spline with no interior knots.

## 6.3 Timing

The time taken for a call of this procedure depends on the complexity of the shape of the data, the value of the smoothing factor $S$, and the number of data points. If the procedure is to be called for different values of $S$, much time can be saved by setting `start = 'W'` after the first call.

It should be noted that choosing $S$ very small considerably increases computation time.

## 6.4 Choice of $S$

If it is known that the product of the weights and $f$-values has a standard deviation approximately equal to some value $\sigma$ for all data points, then $S$ may be chosen to lie in the range $[\sigma^2(m-\sqrt{2m}), \sigma^2(m+\sqrt{2m})]$, where $m$ is the total number of data points. This choice, due to Reinsch [12], is likely to give a good start in the search for a satisfactory value. Otherwise, experimenting with different values of $S$ will be required from the start. In that case, in view of computation time and memory requirements, it is recommended to start with a value of $S$ which is so large that no interior knots are inserted ($p = 8$, $q = 8$). The spline generated in this case is the least-squares bicubic polynomial, and the value returned for `theta`, call it $\theta_0$, gives an upper bound for $S$. Then progressively decrease the value of $S$ to obtain closer fits, say by a factor of 10 in the beginning, i.e., $S = \theta_0/10$, $S = \theta_0/100$, and so on, and more carefully as the approximation shows more details.

Choosing $S$ very small is strongly discouraged. This considerably increases computation time and memory requirements. It may also cause rank deficiency (as indicated by the optional output argument `rank`) and endanger numerical stability.

The number of knots of the spline returned, and their location, generally depend on the value of $S$ and on the behaviour of the function underlying the data. If this procedure is called with a warm start, however, the knots returned may also depend on the smoothing factors of the previous calls. Therefore if, after a number of trials with different values of $S$ and warm starts, a fit can finally be accepted as satisfactory, it may be worthwhile to call the procedure once more with the selected value for $S$ but now using `start = 'C'`. Often, this procedure then returns an approximation with the same quality of fit but with fewer knots, which is therefore better if data reduction is also important.

## 6.5 Weighting of Data Points

The interface for data points lying on a grid does not allow individual weighting of the data values. If these values were determined to widely differing accuracies, it may be better to use the alternative interface for scattered data, which allows non-unit weights. The computation time would generally be very much longer, however.

# Procedure: nag_spline_2d_lsq_fit

## 1 Description

nag_spline_2d_lsq_fit determines a bicubic spline fit $s(x, y)$ to the set of data points $(x_i, y_i, f_i)$ with weights $w_i$, for $i = 1, 2, \ldots, m$. It is defined for $(x, y) \in [a, b] \times [c, d]$, where $a = \min_i x_i$, $b = \max_i x_i$, $c = \min_i y_i$ and $d = \max_i y_i$. The spline has a total of $p$ knots in the $x$-direction, of which you must specify the $p - 8$ interior knots $\lambda_5, \lambda_6, \ldots, \lambda_{p-4}$. Similarly, there are a total of $q$ knots in the $y$-direction, of which you must specify the interior knots $\mu_5, \mu_6, \ldots, \mu_{q-4}$. The knots can be thought of as dividing the data region of the $(x, y)$ plane into panels. A bicubic spline consists of a separate bicubic polynomial in each panel, the polynomials joining together with continuity up to the second derivative across the panel boundaries.

$s(x, y)$ has the property that it minimizes the sum of squares

$$\theta = \sum_{i=1}^{m} \epsilon_i^2$$

of weighted residuals $\epsilon_i = w_i(s(x_i, y_i) - f_i)$, for $i = 1, 2, \ldots, m$, over all bicubic splines with the given knot sets. The procedure produces this minimized value of $\theta$ and the coefficients $\kappa_{ij}$ in the B-spline representation of $s(x, y)$.

The least-squares criterion is not always sufficient to determine the bicubic spline uniquely: there may be a whole family of splines which have the same minimum sum of squares. In these cases, the procedure selects from this family the spline for which the sum of squares of the coefficients $\kappa_{ij}$ is smallest: in other words, the minimal least-squares solution. This choice, although arbitrary, reduces the risk of unwanted fluctuations in the spline fit.

## 2 Usage

```
USE nag_spline_2d
CALL nag_spline_2d_lsq_fit(x, y, f, lambda, mu, spline  [, 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 values of the following problem parameters from the shape of the supplied arrays.

$m \geq 2$ — the number of data points
$p \geq 8$ — the total number of knots in the $x$-direction
$q \geq 8$ — the total number of knots in the $y$-direction

### 3.1 Mandatory Arguments

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

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

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

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

**lambda**($p - 8$) — real(kind=*wp*), intent(in)

*Input:* `lambda(`$i$`)` must contain the interior knot $\lambda_{i+4}$, associated with the variable $x$, for $i = 1, 2, \ldots, p - 8$.

*Note:* a knot is a value of $x$ at which the spline is allowed to be discontinuous in the third derivative with respect to $x$, though continuous up to the second derivative. You can reduce this degree of continuity, if required, by the use of coincident knots, provided that no more than four knots are chosen to coincide at any point. Two, or three coincident knots allow loss of continuity in, respectively, the second and first derivative with respect to $x$ at the value of $x$ at which they coincide. Four coincident knots split the spline surface into two independent parts. For additional advice on the choice of knots see the Further Details section of this module document.

*Constraints:* the knots must be in non-decreasing order, lie strictly within the range covered by the data values of $x$, and have multiplicity $\leq 4$.

**mu**($q - 8$) — real(kind=*wp*), intent(in)

*Input:* `mu(`$j$`)` must contain the interior knot $\mu_{j+4}$, associated with the variable $y$, for $j = 1, 2, \ldots, q - 8$.

*Note:* the same remarks apply to `mu` as to `lambda` above, with $y$ replacing $x$.

*Constraints:* the knots must be in non-decreasing order, lie strictly within the range covered by the data values of $y$, and have multiplicity $\leq 4$.

**spline** — type(nag_spline_2d_comm_*wp*), intent(out)

*Output:* a structure containing details of the spline $s(x, y)$ generated. This structure may be passed to the procedure `nag_spline_2d_eval` to evaluate the spline at given points, or to `nag_spline_2d_intg` to compute its definite integral.

*Note:* to reduce the risk of corrupting the data accidentally, the components of this structure are private; details of the spline may be extracted by calling `nag_spline_2d_extract`.

The procedure allocates roughly $pq$ real(kind=*wp*) 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.

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

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

*Note:* Section 3.3 of the Chapter Introduction gives advice on the choice of weights.

*Default:* `wt` = 1.0.

*Constraints:* at least one element of `wt` must be non-zero.

**theta** — real(kind=*wp*), intent(out), optional

*Output:* the sum of squares of weighted residuals $\theta$.

**diag**($(p - 4)(q - 4)$) — real(kind=*wp*), intent(out), optional

*Output:* `diag` gives the squares of the diagonal elements of the reduced triangular matrix, divided by the mean squared weight. It includes those elements, less than $\epsilon$, which are treated as zero (see Section 6.1).

**thresh** — real(kind=*wp*), intent(in), optional

*Input:* a threshold $\epsilon$ for determining the effective rank of the system of linear equations. The rank is determined as the number of elements of the array `diag` which are considered to be non-zero. An element of `diag` is regarded as zero if it is less than $\epsilon$. The default value `EPSILON(1.0_wp)` is a suitable value for $\epsilon$ in most practical applications which have only 2 or 3 decimals accurate in data. If some coefficients of the fit prove to be very large compared with the data ordinates, this suggests that $\epsilon$ should be increased so as to decrease the rank. The array `diag` will give a guide to appropriate values of $\epsilon$ to achieve this, as well as to the choice of $\epsilon$ in other cases where some experimentation may be needed to determine a value which leads to a satisfactory fit.

*Default:* `thresh` = `EPSILON(1.0_wp)`.

*Constraints:* `thresh` $> 0.0$.

**rank** — integer, intent(out), optional

*Output:* the rank of the system as determined by the value of the threshold $\epsilon$.

*Note:* when `rank` $= (p-4)(q-4)$, the least-squares solution is unique; in other cases the minimal least-squares solution is computed.

**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. |

## Failures (error%level = 2):

| error%code | Description |
|---|---|
| **201** | A solution does not exist.<br>The rank of the system was determined as zero. |

# 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 B-spline representation of the bicubic spline is

$$s(x,y) = \sum_{i=1}^{p-4} \sum_{j=1}^{q-4} \kappa_{ij} M_i(x) N_j(y).$$

Here $M_i(x)$ and $N_j(y)$ denote normalised cubic B-splines, the former defined on the knots $\lambda_i, \lambda_{i+1}, \ldots, \lambda_{i+4}$ and the latter on the knots $\mu_j, \mu_{j+1}, \ldots, \mu_{j+4}$. For further details, see Hayes [11] for bicubic splines and De Boor [5] for normalised B-splines.

The method employed involves forming a system of $m$ linear equations in the coefficients $\kappa_{ij}$ and then computing its least-squares solution, which will be the minimal least-squares solution when appropriate. The basis of the method is described in Hayes [11]. The matrix of the equation is formed using a recurrence relation for B-splines which is numerically stable (see Cox [2] and De Boor [5]; the former contains the more elementary derivation but, unlike De Boor [5], does not cover the case of coincident knots). The least-squares solution is also obtained in a stable manner by using orthogonal transformations, namely a variant of Givens rotation (see Gentleman [10]). This requires only one row of the matrix to be stored at a time. Advantage is taken of the stepped-band structure which the matrix possesses when the data points are suitably ordered, there being at most sixteen non-zero elements in any row because of the definition of B-splines. First the matrix is reduced to upper triangular form and then the diagonal elements of this triangle are examined in turn. When an element is encountered whose square, divided by the mean squared weight, is less than a threshold $\epsilon$, it is replaced by zero and the rest of the elements in its row are reduced to zero by rotations with the remaining rows. The rank of the system is taken to be the number of non-zero diagonal elements in the final triangle, and the non-zero rows of this triangle are used to compute the minimal least-squares solution. If all the diagonal elements are non-zero, the rank is equal to the number of coefficients $\kappa_{ij}$ and the solution obtained is the ordinary least-squares solution, which is unique in this case.

The fit obtained is not defined outside the rectangle

$$a = \min_i x_i \leq x \leq \max_i x_i = b, \quad c = \min_i y_i \leq y \leq \max_i y_i = d.$$

The reason for taking the extreme data values of $x$ and $y$ for the boundary knots is that, as is usual in data fitting, the fit cannot be expected to give satisfactory values outside the data region. If, nevertheless, you require values over a larger rectangle, this can be achieved by augmenting the data with two artificial data points $(a, c, 0)$ and $(b, d, 0)$ with zero weight, where $a \leq x \leq b$, $c \leq y \leq d$ defines the enlarged rectangle. In the case when the data are adequate to make the least-squares solution unique (`rank` $= (p-4)(q-4)$), this enlargement will not affect the fit over the original rectangle, except for possibly enlarged rounding errors, and will simply continue the bicubic polynomials in the panels bordering the rectangle out to the new boundaries; in other cases the fit will be affected. Even using the original rectangle there may be regions within it, particularly at its corners, which lie outside the data region and where, therefore, the fit will be unreliable.

## 6.2 Accuracy

The computation of the B-splines and reduction of the observation matrix to triangular form are both numerically stable.

## 6.3 Timing

The time taken by this procedure is approximately proportional to the number of data points, $m$, and to $(3(q-4)+4)^2$.

## 6.4 Choice of Knots

The choice of the interior knots, which help to determine the spline's shape, must largely be a matter of trial and error. It is usually best to start with a small number of knots and, examining the fit at each stage, add a few knots at a time at places where the fit is particularly poor. In intervals of $x$ or $y$ where the surface represented by the data changes rapidly, in function value or derivatives, more knots will be needed than elsewhere. In some cases guidance can be obtained by analogy with the case of coincident knots: for example, just as three coincident knots can produce a discontinuity in slope, three close knots can produce rapid change in slope. Of course, such rapid changes in behaviour must be adequately represented by the data points, as indeed must the behaviour of the surface generally, if a satisfactory fit is to be achieved. When there is no rapid change in behaviour, equally spaced knots will often suffice.

In all cases the fit should be examined graphically before it is accepted as satisfactory.

# Procedure: nag_spline_2d_interp

## 1 Description

nag_spline_2d_interp determines a bicubic spline $s(x, y)$ which interpolates (passes exactly through) the set of data points $(x_j, y_k, f_{jk})$, for $j = 1, 2, \ldots, m_x$, $k = 1, 2, \ldots, m_y$. The data points are assumed to be ordered such that $x_1 < x_2 < \cdots < x_{m_x}$ and $y_1 < y_2 < \cdots < y_{m_y}$, and the spline is defined for $(x, y) \in [a, b] \times [c, d]$, where $a = x_1$, $b = x_{m_x}$, $c = y_1$ and $d = y_{m_y}$.

## 2 Usage

```
USE nag_spline_2d
CALL nag_spline_2d_interp(x, y, f, spline  [, 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 values of the following problem parameters from the shape of the supplied arrays.

$m_x \geq 4$ — the number of data points along the $x$-axis
$m_y \geq 4$ — the number of data points along the $y$-axis

### 3.1 Mandatory Arguments

**x**($m_x$) — real(kind=*wp*), intent(in)
*Input:* the values $x_j$, for $j = 1, 2, \ldots, m_x$.
*Constraints:* x(1) $<$ x(2) $< \cdots <$ x($m_x$).

**y**($m_y$) — real(kind=*wp*), intent(in)
*Input:* the values $y_k$, for $k = 1, 2, \ldots, m_y$.
*Constraints:* y(1) $<$ y(2) $< \cdots <$ y($m_y$).

**f**($m_x, m_y$) — real(kind=*wp*), intent(in)
*Input:* the values $f_{jk}$ for $j = 1, 2, \ldots, m_x$ and $k = 1, 2, \ldots, m_y$.

**spline** — type(nag_spline_2d_comm_*wp*), intent(out)
*Output:* a structure containing details of the spline $s(x, y)$ generated. This structure may be passed to the procedure nag_spline_2d_eval to evaluate the spline at given points, or to nag_spline_2d_intg to compute its definite integral.
*Note:* to reduce the risk of corrupting the data accidentally, the components of this structure are private; details of the spline may be extracted by calling nag_spline_2d_extract.
The procedure allocates roughly $(m_x+4)(m_y+4)$ real(kind=*wp*) 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 |
|---|---|
| **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. |

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

| error%code | Description |
|---|---|
| **201** | The problem is too ill conditioned to permit solution.<br>A system of linear equations defining the B-spline coefficients was singular. |

## 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

The process of computing the spline consists of the following steps.

1. Choice of the interior $x$-knots $\lambda_5, \lambda_6, \ldots, \lambda_{m_x}$, as $\lambda_j = x_{j-2}$, for $j = 5, 6, \ldots, m_x$.
2. Formation of the system

   $$A_x E = F,$$

   where $A_x$ is a band matrix of order $m_x$ and bandwidth 4, containing in its $j$th row the values at $x_j$ of the B-splines in $x$, $F$ is the $m_x$ by $m_y$ rectangular matrix of values $f_{jk}$, and $E$ denotes an $m_x$ by $m_y$ rectangular matrix of intermediate coefficients.
3. Use of Gaussian elimination to reduce this system to band triangular form.
4. Solution of this triangular system for $E$.
5. Choice of the interior $y$-knots $\mu_5, \mu_6, \ldots, \mu_{m_y}$, as $\mu_k = y_{k-2}$, for $k = 5, 6, \ldots, m_y$.
6. Formation of the system

   $$A_y C^T = E^T,$$

   where $A_y$ is the counterpart of $A_x$ for the $y$ variable, and $C$ denotes the $m_x$ by $m_y$ rectangular matrix of values of $\kappa_{jk}$.

7. Use of Gaussian elimination to reduce this system to band triangular form.
8. Solution of this triangular system for $C^T$ and hence $C$.

For computational convenience, steps (2) and (3), and likewise steps (6) and (7), are combined so that the formation of $A_x$ and $A_y$ and the reductions to triangular form are carried out one row at a time.

## 6.2 Accuracy

The main sources of rounding errors are in steps (2), (3), (6) and (7) of the algorithm described in Section 6.1. It can be shown (see Cox [3]) that the matrix $A_x$ formed in step (2) has elements differing relatively from their true values by at most a small multiple of 3×`EPSILON(1.0_wp)`. $A_x$ is 'totally positive', and a linear system with such a coefficient matrix can be solved quite safely by elimination without pivoting. Similar comments apply to steps (6) and (7). Thus the complete process is numerically stable.

## 6.3 Timing

The time taken by this procedure is approximately proportional to $m_x m_y$.

# Procedure: nag_spline_2d_eval

## 1 Description

This procedure evaluates a bicubic spline $s(x, y)$ at points lying within its region of definition $[a, b] \times [c, d]$ as determined by the particular procedure used to produce the spline. Details may be found in Section 1 of the relevant procedure document.

`nag_spline_2d_eval` is generic and may be used to evaluate the spline

- 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_spline_2d
CALL nag_spline_2d_eval(spline, u, v, s  [, 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` and `s` must all be scalars. |
| **Scattered points:** | `u`, `v` and `s` must all be rank-1 arrays. |
| **Points on a grid:** | `u` and `v` must be rank-1 arrays, and `s` a rank-2 array. |

## 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

**spline** — type(nag_spline_2d_comm_*wp*), intent(in)

*Input:* a structure containing details of the spline $s(x, y)$ to be evaluated.

*Constraints:* `spline` must be as output from a previous call to `nag_spline_2d_auto_fit`, `nag_spline_2d_lsq_fit`, `nag_spline_2d_interp`, or `nag_spline_2d_set`.

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

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

*Constraints:* all values supplied must lie in the interval $[a, b]$ on which the spline is defined, as determined by its generation procedure.

If `s` is scalar, `u` must be scalar;

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

if `s` is a rank-2 array, `u` must be a rank-1 array, with `SIZE(u)` = `SIZE(s,1)`, and the elements of `u` must be strictly increasing.

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

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

*Constraints:* all values supplied must lie in the interval $[c, d]$ on which the spline is defined, as determined by its generation procedure.

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

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

if `s` is a rank-2 array, `v` must be a rank-1 array, with `SIZE(v)` = `SIZE(s,2)`, and the elements of `v` must be strictly increasing.

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

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

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

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

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

### 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 |
|---|---|
| **301** | An input argument has an invalid value. |
| **303** | Array arguments have inconsistent shapes. |

## 5 Examples of Usage

Complete examples of the use of this procedure appear in Examples 1 and 2 of this module document.

## 6 Further Comments

### 6.1 Algorithmic Detail

The procedure is derived from the procedure B2VRE in Anthony *et al.* [1].

## 6.2 Accuracy

The method used to evaluate the B-splines is numerically stable, in the sense that each computed value of $s$ can be regarded as the value that would have been obtained in exact arithmetic from slightly perturbed B-spline coefficients. See Cox [4] for details.

## 6.3 Timing

Computation time is approximately proportional to the number of evaluation points.

# Procedure: nag_spline_2d_intg

## 1 Description

This function evaluates the definite integral

$$I = \int_{\alpha}^{\beta} \int_{\gamma}^{\delta} s(x, y)\, dy\, dx$$

of a bicubic spline $s(x, y)$.

By default the rectangular region of integration $[\alpha, \beta] \times [\gamma, \delta]$ is taken as the region of definition $[a, b] \times [c, d]$ of $s(x, y)$, as determined by the particular procedure used to produce the spline. Details may be found in Section 1 of the relevant procedure document.

Integration may be performed over non-default rectangular regions lying within $[a, b] \times [c, d]$ by supplying one or more of the optional arguments `alpha`, `beta`, `gamma` and `delta`.

## 2 Usage

```
USE nag_spline_2d
[value =] nag_spline_2d_intg(spline  [, optional arguments])
```

The function result value is a scalar of type real(kind=*wp*).

## 3 Arguments

### 3.1 Mandatory Argument

**spline** — type(nag_spline_2d_comm_*wp*), intent(in)

*Input:* a structure containing details of the spline $s(x, y)$ to be integrated.

*Constraints:* `spline` must be as output from a previous call to `nag_spline_2d_auto_fit`, `nag_spline_2d_lsq_fit`, `nag_spline_2d_interp`, or `nag_spline_2d_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.

**alpha** — real(kind=*wp*), intent(in), optional

**beta** — real(kind=*wp*), intent(in), optional

*Input:* the lower and upper limits $\alpha$ and $\beta$ of the integral with respect to $x$.

*Constraints:* $a \leq$ `alpha` $\leq b$ and $a \leq$ `beta` $\leq b$. See Section 1.

*Note:* it is *not* required that `alpha` $<$ `beta`.

*Default:* `alpha` $= a$, `beta` $= b$.

**gamma** — real(kind=*wp*), intent(in), optional

**delta** — real(kind=*wp*), intent(in), optional

*Input:* the lower and upper limits $\gamma$ and $\delta$ of the integral with respect to $y$.

*Constraints:* $c \leq$ `gamma` $\leq d$ and $c \leq$ `delta` $\leq d$. See Section 1.

*Note:* it is *not* required that `gamma` $<$ `delta`.

*Default:* `gamma` $= c$, `delta` $= d$.

**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. |
| **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

The method employed is described in Section 2.1.2 of Dierckx [9].

## 6.2 Timing

The time taken by this function is approximately proportional to $pq$, where $p$ and $q$ are the total numbers of knots in the $x$-direction and the $y$-direction respectively. If these values are not known they may be determined by calling nag_spline_2d_extract.

# Procedure: nag_spline_2d_set

## 1 Description

This procedure creates a structure of type nag_spline_2d_comm_*wp* containing details of a given bicubic spline

$$s(x,y) = \sum_{i=1}^{p-4}\sum_{j=1}^{q-4} \kappa_{ij} M_i(x) N_j(y), \quad (x,y) \in [a,b] \times [c,d],$$

where $M_i(x)$ is the normalized cubic B-spline defined on the knots $\lambda_i, \ldots, \lambda_{i+4}$. The interior knots $\lambda_5, \cdots, \lambda_{p-4}$ must satisfy

$$a < \lambda_5 \leq \lambda_6 \leq \cdots \leq \lambda_{p-5} \leq \lambda_{p-4} < b,$$

and each interior knot must have maximum multiplicity four. Similarly, $N_j(y)$ is the normalized cubic B-spline defined on the knots $\mu_j, \ldots, \mu_{j+4}$. The interior knots $\mu_5, \cdots, \mu_{q-4}$ must satisfy

$$c < \mu_5 \leq \mu_6 \leq \cdots \leq \mu_{q-5} \leq \mu_{q-4} < d,$$

and each interior knot must have maximum multiplicity four. The knot set is completed by setting

$$\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = a, \quad \lambda_{p-3} = \lambda_{p-2} = \lambda_{p-1} = \lambda_p = b$$
$$\mu_1 = \mu_2 = \mu_3 = \mu_4 = c, \quad \mu_{q-3} = \mu_{q-2} = \mu_{q-1} = \mu_q = d.$$

Note that this procedure cannot be used in conjunction with nag_spline_2d_auto_fit with a warm start.

## 2 Usage

```
USE nag_spline_2d
CALL nag_spline_2d_set(a, b, c, d, lambda, mu, kappa, spline  [, 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 values of the following problem parameters from the shape of the supplied arrays.

$p \geq 8$ — the total number of knots in the $x$-direction
$q \geq 8$ — the total number of knots in the $y$-direction

### 3.1 Mandatory Arguments

**a** — real(kind=*wp*), intent(in)
**b** — real(kind=*wp*), intent(in)
*Input:* the lower and upper limits $a$ and $b$ of the $x$-interval on which the spline is defined.
*Constraints:* a < b.

**c** — real(kind=*wp*), intent(in)
**d** — real(kind=*wp*), intent(in)
*Input:* the lower and upper limits $c$ and $d$ of the $y$-interval on which the spline is defined.
*Constraints:* c < d.

**lambda**($p-8$) — real(kind=*wp*), intent(in)

*Input:* `lambda(`$i$`)` must contain the interior knot $\lambda_{i+4}$, for $i = 1, 2, \ldots, p-8$.

*Constraints:* `a` $<$ `lambda(1)` $\leq \cdots \leq$ `lambda(`$p-8$`)` $<$ `b`, and `lambda(`$i$`)` must not have multiplicity $> 4$.

**mu**($q-8$) — real(kind=*wp*), intent(in)

*Input:* `mu(`$j$`)` must contain the interior knot $\mu_{j+4}$, for $j = 1, 2, \ldots, q-8$.

*Constraints:* `c` $<$ `mu(1)` $\leq \cdots \leq$ `mu(`$q-8$`)` $<$ `d`, and `mu(`$i$`)` must not have multiplicity $> 4$.

**kappa**($p-4, q-4$) — real(kind=*wp*), intent(in)

*Input:* `kappa(`$i, j$`)` must contain the coefficient $\kappa_{ij}$ in the B-spline representation of $s(x, y)$.

**spline** — type(nag_spline_2d_comm_*wp*), intent(out)

*Output:* a structure containing details of the spline $s(x, y)$ generated. It may be passed to `nag_spline_2d_eval` to evaluate the spline at specified points, or to `nag_spline_2d_intg` to compute its definite integral.

*Note:* to reduce the risk of corrupting the data accidentally, the components of this structure are private; details of the spline may be extracted by calling `nag_spline_2d_extract`.

The procedure allocates roughly $pq$ real(kind=*wp*) 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 4 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 |
|---|---|
| **301** | An input argument has an invalid value. |
| **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 a spline with data which has been read from file. The procedure `nag_spline_2d_extract` may be used to extract the data to be written to file.

# Procedure: nag_spline_2d_extract

## 1 Description

Given a structure of type nag_spline_2d_comm_*wp* representing a cubic spline

$$s(x,y) = \sum_{i=1}^{p-4}\sum_{j=1}^{q-4} \kappa_{ij} M_i(x) N_j(y), \quad (x,y) \in [a,b] \times [c,d],$$

this procedure optionally returns the total number of knots $p$ and $q$ in each direction, the parameters $a$, $b$, $c$ and $d$, the interior knots $\lambda_5, \ldots, \lambda_{p-4}$ and $\mu_5, \ldots, \mu_{q-4}$, and the B-spline coefficients $\kappa_{ij}$, for $i = 1, \ldots, p-4$, $j = 1, \ldots, q-4$.

Since the number of knots may not be known prior to a call to this procedure the arguments which return the knots and B-spline coefficients are pointers, which are allocated internally. It is your responsibility to deallocate this storage.

## 2 Usage

```
USE nag_spline_2d
CALL nag_spline_2d_extract(spline  [, optional arguments])
```

## 3 Arguments

### 3.1 Mandatory Argument

**spline** — type(nag_spline_2d_comm_*wp*), intent(in)

*Input:* a structure containing details of the spline $s(x,y)$.

*Constraints:* spline must be as output from a previous call to nag_spline_2d_auto_fit, nag_spline_2d_lsq_fit, nag_spline_2d_interp, or nag_spline_2d_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.

**p** — integer, intent(out), optional

**q** — integer, intent(out), optional

*Output:* the total number of knots $p$ and $q$ in the $x$-direction and the $y$-direction, respectively.

**a** — real(kind=*wp*), intent(out), optional

**b** — real(kind=*wp*), intent(out), optional

*Output:* the parameters $a$ and $b$ determining the region of definition of $s(x,y)$.

**c** — real(kind=*wp*), intent(out), optional

**d** — real(kind=*wp*), intent(out), optional

*Output:* the parameters $c$ and $d$ determining the region of definition of $s(x,y)$.

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

*Output:* lambda($i$) holds the interior knot $\lambda_{i+4}$, for $i = 1, 2, \ldots, p-8$.

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

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

*Output:* `mu(`$j$`)` holds the interior knot $\mu_{j+4}$, for $j = 1, 2, \ldots, q-8$.

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

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

*Output:* `kappa(`$i, j$`)` holds the coefficient $\kappa_{ij}$, for $i = 1, 2, \ldots, p-4$ and $j = 1, 2, \ldots, q-4$.

*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. |

# 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 `spline` in order to write to file. The procedure `nag_spline_2d_set` may then be used to initialize a new spline with this data.

If called with no optional arguments `nag_spline_2d_extract` merely checks that the structure `spline` has been created by one of the spline generation procedures of this module.

# Derived Type: nag_spline_2d_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_spline_2d_comm_dp`. For single precision the name is `nag_spline_2d_comm_sp`. Please read the Users' Note for your implementation to check which precisions are available.

## 1 Description

The derived type `nag_spline_2d_comm_wp` is used to represent a two-dimensional bicubic spline $s(x, y)$, in B-spline series form, as described in the Module Introduction.

The procedures `nag_spline_2d_auto_fit`, `nag_spline_2d_lsq_fit`, `nag_spline_2d_interp` and `nag_spline_2d_set` return structures of this type suitable for passing to `nag_spline_2d_eval`, `nag_spline_2d_intg` and `nag_spline_2d_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 `spline` 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, they deallocate that storage before allocating the storage required for the new call.

The components of this type are private.

## 2 Type Definition

```
type nag_spline_2d_comm_wp
  private
  .
  .
  .
end type nag_spline_2d_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_spline_2d_set` and `nag_spline_2d_extract` may be used to initialize and extract data from structures of the type.

# Example 1: Spline fitting with automatic knot selection

Generate a bicubic spline approximation to a set of data points lying on a rectangular mesh, with automatic knot selection. Use several different values of the smoothing factor $S$. For each value of $S$ find the sum of squared residuals, the total number of knots in each direction, and evaluate the spline on a rectangular mesh.

## 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_spline_2d_ex01

  ! Example Program Text for nag_spline_2d
  ! NAG f190, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_spline_2d, ONLY : nag_spline_2d_comm_wp => nag_spline_2d_comm_dp &
   , nag_spline_2d_auto_fit, nag_spline_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, mx, my, nx, ny, p, q
  REAL (wp) :: dx, dy, smooth, theta, xmax, xmin, ymax, ymin
  CHARACTER (1) :: start
  TYPE (nag_spline_2d_comm_wp) :: spline
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: f(:,:), s(:,:), u(:), v(:), x(:), y(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_spline_2d_ex01'

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

  ALLOCATE (x(mx),y(my),f(mx,my),u(nx),v(ny),s(nx,ny)) ! Allocate storage

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

  start = 'cold start'

  DO

    ! Read in successive values of smooth and generate spline for
    ! each.
    READ (nag_std_in,*,end=20) smooth

    ! Determine the spline approximation.

    CALL nag_spline_2d_auto_fit(start,x,y,f,smooth,spline,p=p,q=q, &
     theta=theta)
```

```
        WRITE (nag_std_out,'(//1X,A,1P,E13.4)') &
         'Calling with smoothing factor =', smooth
        WRITE (nag_std_out,'(1X,A,1P,E13.4)') 'Sum of squared residuals =', &
         theta
        WRITE (nag_std_out,'(1X,A,I8)') &
         'Total number of knots in x-direction =', p
        WRITE (nag_std_out,'(1X,A,I8)') &
         'Total number of knots in y-direction =', q

        ! Evaluate the spline on a rectangular grid 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_spline_2d_eval(spline,u,v,s)

        WRITE (nag_std_out,'(/1X,A)') 'Values of computed spline:'
        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), s(:,j)
        END DO

        start = 'warm start'

      END DO
20    CONTINUE

      DEALLOCATE (x,y,f,u,v,s)     ! Deallocate storage

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

    END PROGRAM nag_spline_2d_ex01
```

# 2 Program Data

```
Example Program Data for nag_spline_2d_ex01
 11    9                                                    : mx, my
 6    0.0   5.0                                             : nx, xmin, xmax
 5    0.0   4.0                                             : ny, ymin, ymax
 0.0000E+00  5.0000E-01  1.0000E+00  1.5000E+00  2.0000E+00
 2.5000E+00  3.0000E+00  3.5000E+00  4.0000E+00  4.5000E+00
 5.0000E+00                                                : End of x
 0.0000E+00  5.0000E-01  1.0000E+00  1.5000E+00  2.0000E+00
 2.5000E+00  3.0000E+00  3.5000E+00  4.0000E+00            : End of y
 0.10000E+01 0.15000E+01 0.20600E+01 0.25700E+01 0.30000E+01
 0.35000E+01 0.40400E+01 0.45000E+01 0.50400E+01 0.55050E+01
 0.60000E+01 0.88758E+00 0.13564E+01 0.17552E+01 0.21240E+01
 0.26427E+01 0.31715E+01 0.35103E+01 0.39391E+01 0.43879E+01
 0.48367E+01 0.52755E+01 0.54030E+00 0.82045E+00 0.10806E+01
 0.13508E+01 0.16309E+01 0.18611E+01 0.20612E+01 0.24314E+01
 0.27515E+01 0.29717E+01 0.32418E+01 0.70737E-01 0.10611E+00
 0.15147E+00 0.17684E+00 0.21221E+00 0.24458E+00 0.28595E+00
 0.31632E+00 0.35369E+00 0.38505E+00 0.42442E+00 -.41515E+00
 -.62422E+00 -.83229E+00 -.10404E+01 -.12484E+01 -.14565E+01
 -.16946E+01 -.18627E+01 -.20707E+01 -.22888E+01 -.24769E+01
 -.80114E+00 -.12317E+01 -.16023E+01 -.20029E+01 -.22034E+01
 -.28640E+01 -.32046E+01 -.36351E+01 -.40057E+01 -.44033E+01
 -.48169E+01 -.97999E+00 -.14850E+01 -.19700E+01 -.24750E+01
 -.29700E+01 -.32650E+01 -.39600E+01 -.44550E+01 -.49700E+01
```

```
-.54450E+01 -.59300E+01 -.93446E+00 -.13047E+01 -.18729E+01
-.23511E+01 -.28094E+01 -.32776E+01 -.37958E+01 -.42141E+01
-.46823E+01 -.51405E+01 -.56387E+01 -.65664E+00 -.98547E+00
-.14073E+01 -.16741E+01 -.19809E+01 -.22878E+01 -.26146E+01
-.29314E+01 -.32382E+01 -.35950E+01 -.39319E+01           : End of f
0.1                                                  : 1st smooth value
0.01                                                 : 2nd smooth value
0.001                                                : 3rd smooth value
```

# 3 Program Results

```
Example Program Results for nag_spline_2d_ex01


Calling with smoothing factor =   1.0000E-01
Sum of squared residuals =   1.0004E-01
Total number of knots in x-direction =      10
Total number of knots in y-direction =      13

Values of computed spline:

          x    0.00    1.00    2.00    3.00    4.00    5.00
     y
    4.00      -0.65   -1.36   -1.99   -2.61   -3.25   -3.93
    3.00      -0.98   -1.97   -2.91   -3.91   -4.97   -5.92
    2.00      -0.42   -0.83   -1.24   -1.66   -2.08   -2.48
    1.00       0.54    1.09    1.61    2.14    2.71    3.24
    0.00       0.99    2.04    3.03    4.01    5.02    6.00


Calling with smoothing factor =   1.0000E-02
Sum of squared residuals =   9.9961E-03
Total number of knots in x-direction =      14
Total number of knots in y-direction =      13

Values of computed spline:

          x    0.00    1.00    2.00    3.00    4.00    5.00
     y
    4.00      -0.65   -1.37   -1.97   -2.61   -3.24   -3.93
    3.00      -0.98   -1.97   -2.97   -3.96   -4.97   -5.93
    2.00      -0.42   -0.83   -1.24   -1.68   -2.08   -2.48
    1.00       0.54    1.08    1.64    2.08    2.74    3.24
    0.00       1.00    2.06    3.00    4.04    5.04    6.00


Calling with smoothing factor =   1.0000E-03
Sum of squared residuals =   1.0000E-03
Total number of knots in x-direction =      15
Total number of knots in y-direction =      13

Values of computed spline:

          x    0.00    1.00    2.00    3.00    4.00    5.00
     y
    4.00      -0.66   -1.41   -1.98   -2.61   -3.24   -3.93
    3.00      -0.98   -1.97   -2.97   -3.96   -4.97   -5.93
    2.00      -0.42   -0.83   -1.24   -1.68   -2.08   -2.48
    1.00       0.54    1.08    1.64    2.07    2.75    3.24
    0.00       1.00    2.06    3.00    4.04    5.04    6.00
```

# Example 2: Least-squares spline fitting

Read a set of data points, weights and interior knot positions, and fit a least-squares bicubic spline to the data points. Use an effective rank threshold of $\epsilon = 10^{-6}$. Evaluate the spline at the data points and output $x$, $y$, $f$, and $s(x, y)$ at each 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_spline_2d_ex02

  ! Example Program Text for nag_spline_2d
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_spline_2d, ONLY : nag_spline_2d_comm_wp => nag_spline_2d_comm_dp &
   , nag_spline_2d_lsq_fit, nag_spline_2d_eval, nag_deallocate
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, m, p, q
  TYPE (nag_spline_2d_comm_wp) :: spline
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: f(:), lambda(:), mu(:), s(:), wt(:), x(:), &
   y(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_spline_2d_ex02'

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

  ALLOCATE (x(m),y(m),f(m),wt(m),s(m),lambda(p-8), &
   mu(q-8))                    ! Allocate storage

  READ (nag_std_in,*) x
  READ (nag_std_in,*) y
  READ (nag_std_in,*) f
  READ (nag_std_in,*) wt
  IF (p>8) READ (nag_std_in,*) lambda
  IF (q>8) READ (nag_std_in,*) mu

  ! Fit bicubic spline to data points.

  CALL nag_spline_2d_lsq_fit(x,y,f,lambda,mu,spline,wt=wt, &
   thresh=1.0E-6_wp)

  ! Evaluate spline at the data points and compare with data
  ! values.

  CALL nag_spline_2d_eval(spline,x,y,s)

  WRITE (nag_std_out,*) '       x          y        Data       Fit'
  DO i = 1, m
```

```
        WRITE (nag_std_out,'(4(1X,F10.4))') x(i), y(i), f(i), s(i)
      END DO

      DEALLOCATE (x,y,f,wt,s,lambda,mu) ! Deallocate storage

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

    END PROGRAM nag_spline_2d_ex02
```

# 2 Program Data

```
Example Program Data for nag_spline_2d_ex02
  30                                                : m
  10     8                                          : p , q
  0.60 -0.95  0.87  0.84  0.17 -0.87  1.00  0.10
  0.24 -0.77  0.32  1.00 -0.63 -0.66  0.93  0.15
  0.99 -0.54  0.44 -0.72  0.63 -0.40  0.20  0.43
  0.28 -0.24  0.86 -0.41 -0.05 -1.00              : End of x
 -0.52 -0.61  0.93  0.09  0.88 -0.70  1.00  1.00
  0.30 -0.77 -0.23 -1.00 -0.26 -0.83  0.22  0.89
 -0.80 -0.88  0.68 -0.14  0.67 -0.90 -0.84  0.84
  0.15 -0.91 -0.35 -0.16 -0.35 -1.00              : End of y
  0.93 -1.79  0.36  0.52  0.49 -1.76  0.33  0.48
  0.65 -1.82  0.92  1.00  8.88 -2.01  0.47  0.49
  0.84 -2.42  0.47  7.15  0.44 -3.34  2.78  0.44
  0.70 -6.52  0.66  2.32  1.66 -1.00              : End of f
  10.0  10.0  10.0  10.0  10.0  10.0  1.00  1.00
  1.00  1.00  1.00  1.00  1.00  1.00  1.00  1.00
  1.00  1.00  1.00  1.00  1.00  1.00  1.00  1.00
  1.00  1.00  1.00  1.00  1.00  1.00              : End of wt
 -0.5   0.0                                        : End of lambda
```

# 3 Program Results

```
 Example Program Results for nag_spline_2d_ex02
        x          y        Data       Fit
     0.6000    -0.5200     0.9300     0.9441
    -0.9500    -0.6100    -1.7900    -1.7931
     0.8700     0.9300     0.3600     0.3529
     0.8400     0.0900     0.5200     0.5024
     0.1700     0.8800     0.4900     0.4705
    -0.8700    -0.7000    -1.7600    -1.7521
     1.0000     1.0000     0.3300     0.6315
     0.1000     1.0000     0.4800     1.4910
     0.2400     0.3000     0.6500     0.9241
    -0.7700    -0.7700    -1.8200    -2.4301
     0.3200    -0.2300     0.9200    -0.3692
     1.0000    -1.0000     1.0000     1.0835
    -0.6300    -0.2600     8.8800     7.6346
    -0.6600    -0.8300    -2.0100    -1.5815
     0.9300     0.2200     0.4700     1.4912
     0.1500     0.8900     0.4900     0.4414
     0.9900    -0.8000     0.8400     0.5495
    -0.5400    -0.8800    -2.4200    -2.6795
     0.4400     0.6800     0.4700     1.5862
    -0.7200    -0.1400     7.1500     7.5708
     0.6300     0.6700     0.4400     0.6288
    -0.4000    -0.9000    -3.3400    -4.6955
     0.2000    -0.8400     2.7800     1.7123
     0.4300     0.8400     0.4400     0.6888
     0.2800     0.1500     0.7000     0.7713
```

```
-0.2400    -0.9100    -6.5200    -4.7072
 0.8600    -0.3500     0.6600     0.9347
-0.4100    -0.1600     2.3200     2.7039
-0.0500    -0.3500     1.6600     2.2865
-1.0000    -1.0000    -1.0000    -1.0228
```

# Example 3: Spline interpolation

Given the set of data points $(x_j, y_k, f_{jk})$, for $j = 1, 2, \ldots, m_x$ and $k = 1, 2, \ldots, m_y$, construct a bicubic spline interpolant. Output the total number of knots in each direction, the interior knots and the B-spline coefficients.

## 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_spline_2d_ex03

  ! Example Program Text for nag_spline_2d
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_spline_2d, ONLY : nag_spline_2d_comm_wp => nag_spline_2d_comm_dp &
   , nag_spline_2d_interp, nag_spline_2d_extract, nag_deallocate
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: mx, my, p, q
  TYPE (nag_spline_2d_comm_wp) :: spline
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: f(:,:), x(:), y(:)
  REAL (wp), POINTER :: kappa(:,:), lambda(:), mu(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_spline_2d_ex03'

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

  ALLOCATE (x(mx),y(my),f(mx,my)) ! Allocate storage

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

  ! Generate the (x,y,f) interpolating bicubic B-spline.

  CALL nag_spline_2d_interp(x,y,f,spline)

  ! Extract the interior knots lambda, mu and the B-spline
  ! coefficients kappa from the structure.

  CALL nag_spline_2d_extract(spline,p=p,q=q,lambda=lambda,mu=mu, &
   kappa=kappa)

  WRITE (nag_std_out,'(/1X,A,I8)') &
   'Total number of knots in x-direction: p =', p
  WRITE (nag_std_out,'(/1X,A)') 'Interior knots in x-direction:'
  WRITE (nag_std_out,'(5ES15.4)') lambda
  WRITE (nag_std_out,'(/1X,A,I8)') &
   'Total number of knots in y-direction: q =', q
```

```
      WRITE (nag_std_out,'(/1X,A)') 'Interior knots in y-direction:'
      WRITE (nag_std_out,'(5ES15.4)') mu

      CALL nag_write_gen_mat(kappa,title= &
       'The matrix kappa of B-spline coefficients:')

      DEALLOCATE (x,y,f,lambda,mu,kappa) ! Deallocate storage

      NULLIFY (lambda,mu,kappa)

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

    END PROGRAM nag_spline_2d_ex03
```

# 2 Program Data

```
Example Program Data for nag_spline_2d_ex03
  7  6                                        : mx , my
  1.00  1.10  1.30  1.50  1.60  1.80  2.00 : End of x
  0.00  0.10  0.40  0.70  0.90  1.00       : End of y
  1.00  1.21  1.69  2.25  2.56  3.24  4.00
  1.10  1.31  1.79  2.35  2.66  3.34  4.10
  1.40  1.61  2.09  2.65  2.96  3.64  4.40
  1.70  1.91  2.39  2.95  3.26  3.94  4.70
  1.90  2.11  2.59  3.15  3.46  4.14  4.90
  2.00  2.21  2.69  3.25  3.56  4.24  5.00 : End of f
```

# 3 Program Results

```
 Example Program Results for nag_spline_2d_ex03

 Total number of knots in x-direction: p =      11

 Interior knots in x-direction:
     1.3000E+00     1.5000E+00     1.6000E+00

 Total number of knots in y-direction: q =      10

 Interior knots in y-direction:
     4.0000E-01     7.0000E-01
 The matrix kappa of B-spline coefficients:
       1.0000     1.1333     1.3667     1.7000     1.9000     2.0000
       1.2000     1.3333     1.5667     1.9000     2.1000     2.2000
       1.5833     1.7167     1.9500     2.2833     2.4833     2.5833
       2.1433     2.2767     2.5100     2.8433     3.0433     3.1433
       2.8667     3.0000     3.2333     3.5667     3.7667     3.8667
       3.4667     3.6000     3.8333     4.1667     4.3667     4.4667
       4.0000     4.1333     4.3667     4.7000     4.9000     5.0000
```

# Example 4: Initializing a spline

Initialize a spline structure with the interior knots and B-spline coefficients generated in Example 3. Compute the definite integral over a subregion of the the region of definition of the spline.

## 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_spline_2d_ex04

  ! Example Program Text for nag_spline_2d
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_spline_2d, ONLY : nag_spline_2d_comm_wp => nag_spline_2d_comm_dp
  USE nag_spline_2d, ONLY : nag_spline_2d_set
  USE nag_spline_2d, ONLY : nag_spline_2d_intg
  USE nag_spline_2d, ONLY : nag_deallocate
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: p, q
  REAL (wp) :: a, alpha, b, beta, c, d, delta, gamma, integral
  TYPE (nag_spline_2d_comm_wp) :: spline
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: kappa(:,:), lambda(:), mu(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_spline_2d_ex04'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) a, b, c, d
  READ (nag_std_in,*) p, q

  ALLOCATE (lambda(p-8),mu(q-8),kappa(p-4,q-4)) ! Allocate storage

  READ (nag_std_in,*) lambda
  READ (nag_std_in,*) mu
  READ (nag_std_in,*) kappa

  ! Initialize spline.

  CALL nag_spline_2d_set(a,b,c,d,lambda,mu,kappa,spline)

  ! Read integration region.
  READ (nag_std_in,*) alpha, beta, gamma, delta

  ! Evaluate spline integral on [alpha, beta] x [gamma,delta].

  integral = nag_spline_2d_intg(spline,alpha=alpha,beta=beta,gamma=gamma, &
   delta=delta)

  WRITE (nag_std_out,'(/1X,A,ES12.4)') 'Spline integral = ', integral

  DEALLOCATE (lambda,mu,kappa) ! Deallocate storage
```

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

    END PROGRAM nag_spline_2d_ex04
```

# 2 Program Data

```
Example Program Data for nag_spline_2d_ex04
 1.0   2.0   0.0   1.0                   : a, b, c, d
 11    10                               : p,  q
 1.3000E+00     1.5000E+00     1.6000E+00 : End of lambda
 4.0000E-01     7.0000E-01                : End of mu
 1.0000E+00     1.2000E+00     1.5833E+00     2.1433E+00     2.8667E+00
 3.4667E+00     4.0000E+00     1.1333E+00     1.3333E+00     1.7167E+00
 2.2767E+00     3.0000E+00     3.6000E+00     4.1333E+00     1.3667E+00
 1.5667E+00     1.9500E+00     2.5100E+00     3.2333E+00     3.8333E+00
 4.3667E+00     1.7000E+00     1.9000E+00     2.2833E+00     2.8433E+00
 3.5667E+00     4.1667E+00     4.7000E+00     1.9000E+00     2.1000E+00
 2.4833E+00     3.0433E+00     3.7667E+00     4.3667E+00     4.9000E+00
 2.0000E+00     2.2000E+00     2.5833E+00     3.1433E+00     3.8667E+00
 4.4667E+00     5.0000E+00                : End of kappa
 1.5   2.0   0.5   1.0                   : alpha, beta, gamma, delta
```

# 3 Program Results

```
 Example Program Results for nag_spline_2d_ex04

 Spline integral =   9.5834E-01
```

# Further Details

## 1 Choice of Knots for nag_spline_2d_lsq_fit

nag_spline_2d_lsq_fit fits to arbitrary data points, with arbitrary weights, a least-squares bicubic spline surface with given interior knots. The choice of these knots so as to give an acceptable fit must largely be a matter of trial and error, though with a little experience a satisfactory choice can often be made after one or two trials. It is usually best to start with a small number of knots (too many will result in unwanted fluctuations in the fit, or even in there being no unique solution) and, examining the fit graphically at each stage, to add a few knots at a time at places where the fit is particularly poor. Moving the existing knots towards these places will also often improve the fit. In regions where the behaviour of the surface underlying the data is changing rapidly, closer knots will be needed than elsewhere. Otherwise, positioning is not usually very critical and equally spaced knots are often satisfactory.

A useful feature of the procedure is that it can be used in applications which require the continuity to be less than the normal continuity of the bicubic spline. For example, the approximant may be required to have a discontinuous partial derivative with respect to $x$, for some value of $x$. This can be achieved by placing three coincident knots in the $x$-direction at the given value of $x$. Similarly a discontinuity in the second partial derivative for some value of $y$ can be achieved by placing two $y$-direction knots there. Analogy with these discontinuous cases can provide guidance in more usual cases: for example, just as three coincident knots can produce a discontinuity in slope, so three close knots can produce a rapid change in slope. The closer the knots are, the more rapid can the change be.

An illustration of knot selection for least-squares spline fitting in *one dimension* appears in the Further details section of the module document for nag_spline_1d (8.2). The guiding principles illustrated there should be applied to the selection of interior knots in each of the two directions for nag_spline_2d_lsq_fit.

# References

[1] Anthony G T, Cox M G and Hayes J G (1982) *DASL – Data Approximation Subroutine Library* National Physical Laboratory

[2] Cox M G (1972) The numerical evaluation of B-splines *J. Inst. Math. Appl.* **10** 134–149

[3] Cox M G (1975) An algorithm for spline interpolation *J. Inst. Math. Appl.* **15** 95–108

[4] Cox M G (1978) The numerical evaluation of a spline from its B-spline representation *J. Inst. Math. Appl.* **21** 135–143

[5] De Boor C (1972) On calculating with B-splines *J. Approx. Theory* **6** 50–62

[6] Dierckx P (1981) An algorithm for surface fitting with spline functions *IMA J. Numer. Anal.* **1** 267–283

[7] Dierckx P (1981) An improved algorithm for curve fitting with spline functions *Report TW54* Department of Computer Science, Katholieke Universiteit Leuven

[8] Dierckx P (1982) A fast algorithm for smoothing data on a rectangular grid while using spline functions *SIAM J. Numer. Anal.* **19** 1286–1304

[9] Dierckx P (1993) *Curve and Surface Fitting with Splines* Clarendon Press, Oxford

[10] Gentleman W M (1969) An error analysis of Goertzel's (Watt's) method for computing Fourier coefficients *Comput. J.* **12** 160–165

[11] Hayes J G (1974) Numerical methods for curve and surface fitting *Bull. Inst. Math. Appl.* **10** 144–152

[12] Reinsch C H (1967) Smoothing by spline functions *Numer. Math.* **10** 177–183