# NAG Library Routine Document

# E02JDF

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

**Note**: *this routine uses* **optional parameters** *to define choices in the problem specification and in the details of the algorithm. If you wish to use* default *settings for all of the optional parameters, you need only read Sections 1 to 9 of this document. If, however, you wish to reset some or all of the settings please refer to Section 10 for a detailed description of the specification of the optional parameters produced by the routine.*

## 1 Purpose

E02JDF computes a spline approximation to a set of scattered data using a two-stage approximation method.

The computational complexity of the method grows linearly with the number of data points; hence large datasets are easily accommodated.

## 2 Specification

```
SUBROUTINE E02JDF (N, X, Y, F, LSMINP, LSMAXP, NXCELS, NYCELS, LCOEFS,          &
                   COEFS, IOPTS, OPTS, IFAIL)

INTEGER            N, LSMINP, LSMAXP, NXCELS, NYCELS, LCOEFS, IOPTS(*),         &
                   IFAIL
REAL (KIND=nag_wp) X(N), Y(N), F(N), COEFS(LCOEFS), OPTS(*)
```

Before calling E02JDF, E02ZKF must be called with OPTSTR set to '**Initialize** = E02JDF'. Settings for optional algorithmic parameters may be specified by calling E02ZKF before a call to E02JDF.

## 3 Description

E02JDF determines a smooth bivariate spline approximation to a set of data points $(x_i, y_i, f_i)$, for $i = 1, 2, \ldots, n$. Here, 'smooth' means $C^1$.

The approximation domain is the bounding box $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$, where $x_{\min}$ (respectively $y_{\min}$) and $x_{\max}$ (respectively $y_{\max}$) denote the lowest and highest data values of the $(x_i)$ (respectively $(y_i)$).

The spline is computed by local approximations on a uniform triangulation of the bounding box. These approximations are extended to a smooth spline representation of the surface over the domain. The local approximation scheme is by least-squares polynomials (Davydov and Zeilfelder (2004)).

The two-stage approximation method employed by E02JDF is derived from the TSFIT package of O. Davydov and F. Zeilfelder.

Values of the computed spline can subsequently be computed by calling E02JEF or E02JFF.

## 4 References

Davydov O and Zeilfelder F (2004) Scattered data fitting by direct extension of local polynomials to bivariate splines *Advances in Comp. Math.* **21** 223–271

# 5 Parameters

1: N – INTEGER *Input*

*On entry*: $n$, the number of data values to be fitted.

*Constraint*: N $> 1$.

2: X(N) – REAL (KIND=nag_wp) array *Input*
3: Y(N) – REAL (KIND=nag_wp) array *Input*
4: F(N) – REAL (KIND=nag_wp) array *Input*

*On entry*: the $(x_i, y_i, f_i)$ data values to be fitted.

*Constraint*: X$(j) \neq$ X$(1)$ for some $j = 2, \ldots, n$ and Y$(k) \neq$ Y$(1)$ for some $k = 2, \ldots, n$; i.e., there are at least two distinct $x$ and $y$ values.

5: LSMINP – INTEGER *Input*
6: LSMAXP – INTEGER *Input*

*On entry*: are control parameters for the local approximations.

Each local approximation is computed on a local domain containing one of the triangles in the discretization of the bounding box. The size of each local domain will be adaptively chosen such that if it contains fewer than LSMINP sample points it is expanded, else if it contains greater than LSMAXP sample points a thinning method is applied. LSMAXP mainly controls computational cost (in that working with a thinned set of points is cheaper and may be appropriate if the input data is densely distributed), while LSMINP allows handling of different types of scattered data.

Setting LSMAXP $<$ LSMINP, and therefore forcing either expansion or thinning, may be useful for computing initial coarse approximations. In general smaller values for these arguments reduces cost.

A calibration procedure (experimenting with a small subset of the data to be fitted and validating the results) may be needed to choose the most appropriate values for LSMINP and LSMAXP.

*Constraints*:

$1 \leq$ LSMINP $\leq$ N;
LSMAXP $\geq 1$.

7: NXCELS – INTEGER *Input*
8: NYCELS – INTEGER *Input*

*On entry*: NXCELS (respectively NYCELS) is the number of cells in the $x$ (respectively $y$) direction that will be used to create the triangulation of the bounding box of the domain of the function to be fitted.

Greater efficiency generally comes when NXCELS and NYCELS are chosen to be of the same order of magnitude and are such that N is $O$(NXCELS $\times$ NYCELS). Thus for a 'square' triangulation — when NXCELS $=$ NYCELS — the quantities $\sqrt{\text{N}}$ and NXCELS should be of the same order of magnitude. See also Section 8.

*Constraints*:

NXCELS $\geq 1$;
NYCELS $\geq 1$.

9: LCOEFS – INTEGER *Input*
10: COEFS(LCOEFS) – REAL (KIND=nag_wp) array *Output*

*On exit*: if IFAIL $= 0$ on exit, COEFS contains the computed spline coefficients.

*Constraint*: LCOEFS $\geq$ (((NXCELS $+ 2$) $\times$ (NYCELS $+ 2$) $+ 1$)/2) $\times 10 + 1$.

11: IOPTS(∗) – INTEGER array *Communication Array*

*On entry*: the contents of IOPTS **must not** be modified in any way either directly or indirectly, by further calls to E02ZKF, before calling either or both of the evaluation routines E02JEF and E02JFF.

12: OPTS(∗) – REAL (KIND=nag_wp) array *Communication Array*

*On entry*: the contents of OPTS **must not** be modified in any way either directly or indirectly, by further calls to E02ZKF, before calling either or both of the evaluation routines E02JEF and E02JFF.

13: IFAIL – INTEGER *Input/Output*

*On entry*: IFAIL must be set to 0, $-1$ or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value $-1$ or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this parameter, the recommended value is 0. **When the value $-1$ or 1 is used it is essential to test the value of IFAIL on exit.**

*On exit*: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

## 6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or $-1$, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 2$

On entry, N $= \langle value \rangle$.
Constraint: N $> 1$.

IFAIL $= 4$

On entry, LSMINP $= \langle value \rangle$ and N $= \langle value \rangle$.
Constraint: $1 \leq$ LSMINP $\leq$ N.

IFAIL $= 5$

On entry, LSMAXP $= \langle value \rangle$.
Constraint: LSMAXP $\geq 1$.

IFAIL $= 6$

On entry, NXCELS $= \langle value \rangle$.
Constraint: NXCELS $\geq 1$.

IFAIL $= 7$

On entry, NYCELS $= \langle value \rangle$.
Constraint: NYCELS $\geq 1$.

IFAIL $= 8$

On entry, LCOEFS $= \langle value \rangle$.
Constraint: LCOEFS $\geq (((\text{NXCELS} + 2) \times (\text{NYCELS} + 2) + 1)/2) \times 10 + 1$.

IFAIL $= 9$

Option arrays are not initialized or are corrupted.

IFAIL $= 11$

An unexpected algorithmic failure was encountered. Please contact NAG.

IFAIL $= 12$

On entry, all elements of X or of Y are equal.

IFAIL $= 21$

The value of optional parameter **Polynomial Starting Degree** was invalid.

IFAIL $= -999$

Dynamic memory allocation failed.

## 7 Accuracy

Technical results on error bounds can be found in Davydov and Zeilfelder (2004).

Local approximation by polynomials of degree $d$ for $n$ data points has optimal approximation order $n^{-(d+1)/2}$.

The approximation error for $C^1$ global smoothing is $O\left(n^{-2}\right)$.

Whether maximal accuracy is achieved depends on the distribution of the input data and the choices of the algorithmic parameters. The reference above contains extensive numerical tests and further technical discussions of how best to configure the method.

## 8 Further Comments

$n$-linear complexity and memory usage can be attained for sufficiently dense input data if the triangulation parameters NXCELS and NYCELS are chosen as recommended in their descriptions above. For sparse input data on such triangulations, if many expansion steps are required (see LSMINP) the complexity may rise to be loglinear.

## 9 Example

The Franke function

$$\begin{aligned} f(x,y) \;=\; & 0.75\exp\left(-\left((9x-2)^2+(9y-2)^2\right)/4\right)+ \\ & 0.75\exp\left(-(9x+1)^2/49-(9y+1)/10\right)+ \\ & 0.5\exp\left(-\left((9x-7)^2+(9y-3)^2\right)/4\right)- \\ & 0.2\exp\left(-(9x-4)^2-(9y-7)^2\right) \end{aligned}$$

is widely used for testing surface-fitting methods. The example program randomly generates a number of points on this surface. From these a spline is computed and then evaluated at a vector of points and on a mesh.

### 9.1 Program Text

```
    Program e02jdfe

!     E02JDF Example Program Text

!     Mark 24 Release. NAG Copyright 2012.

!     .. Use Statements ..
      Use nag_library, Only: e02jdf, nag_wp
!     .. Implicit None Statement ..
      Implicit None
```

```
!     .. Parameters ..
      Integer, Parameter                  :: liopts = 100, lopts = 100,       &
                                             nin = 5, nout = 6
!     .. Local Scalars ..
      Integer                             :: ifail, lcoefs, lsmaxp, lsminp,   &
                                             n, nxcels, nycels
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable     :: coefs(:), f(:), x(:), y(:)
      Real (Kind=nag_wp)                  :: opts(lopts), pmax(2), pmin(2)
      Integer                             :: iopts(liopts)
!     .. Intrinsic Procedures ..
      Intrinsic                           :: maxval, minval
!     .. Executable Statements ..
      Write (nout,*) 'E02JDF Example Program Results'

!     Generate the data to fit.

      Call generate_data(n,x,y,f,lsminp,lsmaxp,nxcels,nycels,lcoefs,coefs)

!     Initialize the options arrays and set/get some options.

      Call handle_options(iopts,liopts,opts,lopts)

!     Compute the spline coefficients.

      ifail = 0
      Call e02jdf(n,x,y,f,lsminp,lsmaxp,nxcels,nycels,lcoefs,coefs,iopts,opts, &
        ifail)

!     pmin and pmax form the bounding box of the spline. We must not attempt to
!     evaluate the spline outside this box.

      pmin(:) = (/minval(x),minval(y)/)
      pmax(:) = (/maxval(x),maxval(y)/)

      Deallocate (x,y,f)

!     Evaluate the approximation at a vector of values.

      Call evaluate_at_vector(coefs,iopts,opts,pmin,pmax)

!     Evaluate the approximation on a mesh.

      Call evaluate_on_mesh(coefs,iopts,opts,pmin,pmax)

    Contains
      Subroutine generate_data(n,x,y,f,lsminp,lsmaxp,nxcels,nycels,lcoefs, &
        coefs)

!       Reads n from a data file and then generates an x and a y vector of n
!       pseudorandom uniformly distributed values on (0,1]. These are passed
!       to the bivariate function of R. Franke to create the data set to fit.
!       The remaining input data for E02JDF are set to suitable values for
!       this problem, as discussed by Davydov and Zeilfelder.

!       .. Use Statements ..
        Use nag_library, Only: g05kff, g05saf
!       .. Implicit None Statement ..
        Implicit None
!       .. Parameters ..
        Integer, Parameter                  :: lseed = 1, mstate = 17
!       .. Scalar Arguments ..
        Integer, Intent (Out)               :: lcoefs, lsmaxp, lsminp, n,     &
                                               nxcels, nycels
!       .. Array Arguments ..
        Real (Kind=nag_wp), Allocatable, Intent (Out) :: coefs(:), f(:), x(:), &
                                                         y(:)
!       .. Local Scalars ..
        Integer                             :: ifail, lstate, subid
!       .. Local Arrays ..
        Integer                             :: seed(lseed), state(mstate)
```

```
!       .. Intrinsic Procedures ..
        Intrinsic                           :: exp
!       .. Executable Statements ..
        Continue

!       Read the size of the data set to be generated and fitted.
!       (Skip the heading in the data file.)

        Read (nin,*)
        Read (nin,*) n
        Allocate (x(n),y(n),f(n))

!       Initialize the random number generator and then generate the data.

        seed(:) = (/32958/)
        lstate = mstate

        ifail = 0
        Call g05kff(1,subid,seed,lseed,state,lstate,ifail)

        ifail = 0
        Call g05saf(n,state,x,ifail)

        ifail = 0
        Call g05saf(n,state,y,ifail)

!       Ensure that the bounding box stretches all the way to (0,0) and (1,1)
        x(1) = 0.0_nag_wp
        y(1) = 0.0_nag_wp
        x(n) = 1.0_nag_wp
        y(n) = 1.0_nag_wp

        f(:) = 0.75_nag_wp*exp(-((9._nag_wp*x(:)-2._nag_wp)**2+(9._nag_wp*y(:) &
          -2._nag_wp)**2)/4._nag_wp) + 0.75_nag_wp*exp(-(9._nag_wp*x(:)+ &
          1._nag_wp)**2/49._nag_wp-(9._nag_wp*y(:)+1._nag_wp)/10._nag_wp) + &
          0.5_nag_wp*exp(-((9._nag_wp*x(:)-7._nag_wp)**2+(9._nag_wp*y(:)- &
          3._nag_wp)**2)/4._nag_wp) - 0.2_nag_wp*exp(-(9._nag_wp*x(:)- &
          4._nag_wp)**2-(9._nag_wp*y(:)-7._nag_wp)**2)

!       Grid size for the approximation.

        nxcels = 6
        nycels = nxcels

!       Identify the computation.

        Write (nout,*)
        Write (nout,*) 'Computing the coefficients of a C^1 spline &
          &approximation to Franke''s function'
        Write (nout,99999) 'Using a ', nxcels, ' by ', nycels, ' grid'

!       Local-approximation control parameters.

        lsminp = 3
        lsmaxp = 100

!       Set up the array to hold the computed spline coefficients.

        lcoefs = (((nxcels+2)*(nycels+2)+1)/2)*10 + 1
        Allocate (coefs(lcoefs))

        Return
99999   Format (1X,A,I2,A,I2,A)
      End Subroutine generate_data
      Subroutine handle_options(iopts,liopts,opts,lopts)

!       Auxiliary routine for initializing the options arrays and
!       for demonstrating how to set and get optional parameters.

!       .. Use Statements ..
        Use nag_library, Only: e02zkf, e02zlf
```

```
!       .. Scalar Arguments ..
        Integer, Intent (In)                 :: liopts, lopts
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: opts(lopts)
        Integer, Intent (Out)                :: iopts(liopts)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                   :: rvalue
        Integer                              :: ifail, ivalue, optype
        Character (3)                        :: cvalue
        Character (80)                       :: optstr
!       .. Executable Statements ..
        ifail = 0
        Call e02zkf('Initialize = E02JDF',iopts,liopts,opts,lopts,ifail)

!       Set some non-default parameters for the local approximation method.

        Write (optstr,99999) 'Minimum Singular Value LPA = ', &
          1._nag_wp/32._nag_wp

        ifail = 0
        Call e02zkf(optstr,iopts,liopts,opts,lopts,ifail)

        ifail = 0
        Call e02zkf('Polynomial Starting Degree = 3',iopts,liopts,opts,lopts, &
          ifail)

!       Set some non-default parameters for the global approximation method.

        ifail = 0
        Call e02zkf('Averaged Spline = Yes',iopts,liopts,opts,lopts,ifail)

!       As an example of how to get the value of an optional parameter,
!       display whether averaging of local approximations is in operation.

        ifail = 0
        Call e02zlf('Averaged Spline',ivalue,rvalue,cvalue,optype,iopts,opts, &
          ifail)

        If (cvalue=='YES') Then
          Write (nout,*) 'Using an averaged local approximation'
        End If

        Return
99999   Format (A,E16.9)
      End Subroutine handle_options
      Subroutine evaluate_at_vector(coefs,iopts,opts,pmin,pmax)

!       Evaluates the approximation at a vector of values.

!       .. Use Statements ..
        Use nag_library, Only: e02jef
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: coefs(*), opts(*), pmax(2),    &
                                                pmin(2)
        Integer, Intent (In)                 :: iopts(*)
!       .. Local Scalars ..
        Integer                              :: i, ifail, nevalv
!       .. Local Arrays ..
        Real (Kind=nag_wp), Allocatable      :: fevalv(:), xevalv(:), yevalv(:)
!       .. Intrinsic Procedures ..
        Intrinsic                            :: max, min
!       .. Executable Statements ..
        Read (nin,*) nevalv
        Allocate (xevalv(nevalv),yevalv(nevalv),fevalv(nevalv))

        Read (nin,*)(xevalv(i),yevalv(i),i=1,nevalv)

!       Force the points to be within the bounding box of the spline.

        Do i = 1, nevalv
          xevalv(i) = max(xevalv(i),pmin(1))
```

```
          xevalv(i) = min(xevalv(i),pmax(1))
          yevalv(i) = max(yevalv(i),pmin(2))
          yevalv(i) = min(yevalv(i),pmax(2))
        End Do

        ifail = 0
        Call e02jef(nevalv,xevalv,yevalv,coefs,fevalv,iopts,opts,ifail)

        Write (nout,*)
        Write (nout,*) 'Values of computed spline at (x_i,y_i):'
        Write (nout,*)
        Write (nout,99999) 'x_i', 'y_i', 'f(x_i,y_i)'
        Write (nout,99998)(xevalv(i),yevalv(i),fevalv(i),i=1,nevalv)

        Return
99999   Format (1X,3A12)
99998   Format (1X,3F12.2)
      End Subroutine evaluate_at_vector
      Subroutine evaluate_on_mesh(coefs,iopts,opts,pmin,pmax)

!       Evaluates the approximation on a mesh of n_x * n_y values.

!       .. Use Statements ..
        Use nag_library, Only: e02jff
!       .. Implicit None Statement ..
        Implicit None
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: coefs(*), opts(*), pmax(2),    &
                                                pmin(2)
        Integer, Intent (In)                 :: iopts(*)
!       .. Local Scalars ..
        Integer                              :: i, ifail, j, nxeval, nyeval
        Logical                              :: print_mesh
!       .. Local Arrays ..
        Real (Kind=nag_wp), Allocatable      :: fevalm(:,:), xevalm(:),        &
                                                yevalm(:)
        Real (Kind=nag_wp)                   :: h(2), ll_corner(2), ur_corner(2)
!       .. Intrinsic Procedures ..
        Intrinsic                            :: max, min, real
!       .. Executable Statements ..
        Read (nin,*) nxeval, nyeval
        Allocate (xevalm(nxeval),yevalm(nyeval),fevalm(nxeval,nyeval))

!       Define the mesh by its lower-left and upper-right corners.

        Read (nin,*) ll_corner(1:2)
        Read (nin,*) ur_corner(1:2)

!       Set the mesh spacing and the evaluation points.
!       Force the points to be within the bounding box of the spline.

        h(1) = (ur_corner(1)-ll_corner(1))/real(nxeval-1,nag_wp)
        h(2) = (ur_corner(2)-ll_corner(2))/real(nyeval-1,nag_wp)

        Do i = 1, nxeval
          xevalm(i) = ll_corner(1) + real(i-1,nag_wp)*h(1)
          xevalm(i) = max(xevalm(i),pmin(1))
          xevalm(i) = min(xevalm(i),pmax(1))
        End Do

        Do j = 1, nyeval
          yevalm(j) = ll_corner(2) + real(j-1,nag_wp)*h(2)
          yevalm(j) = max(yevalm(j),pmin(2))
          yevalm(j) = min(yevalm(j),pmax(2))
        End Do

!       Evaluate.

        ifail = 0
        Call e02jff(nxeval,nyeval,xevalm,yevalm,coefs,fevalm,iopts,opts,ifail)
```

```
!       Output the computed function values?

        Read (nin,*) print_mesh

        If (.Not. print_mesh) Then
          Write (nout,*)
          Write (nout,*) &
            'Outputting of the function values on the mesh is disabled'
        Else
          Write (nout,*)
          Write (nout,*) 'Values of computed spline at (x_i,y_j):'
          Write (nout,*)
          Write (nout,99999) 'x_i', 'y_j', 'f(x_i,y_j)'
          Write (nout,99998)((xevalm(i),yevalm(j),fevalm(i, &
            j),i=1,nxeval),j=1,nyeval)
        End If

        Return
99999   Format (1X,3A12)
99998   Format (1X,3F12.2)
      End Subroutine evaluate_on_mesh
    End Program e02jdfe
```

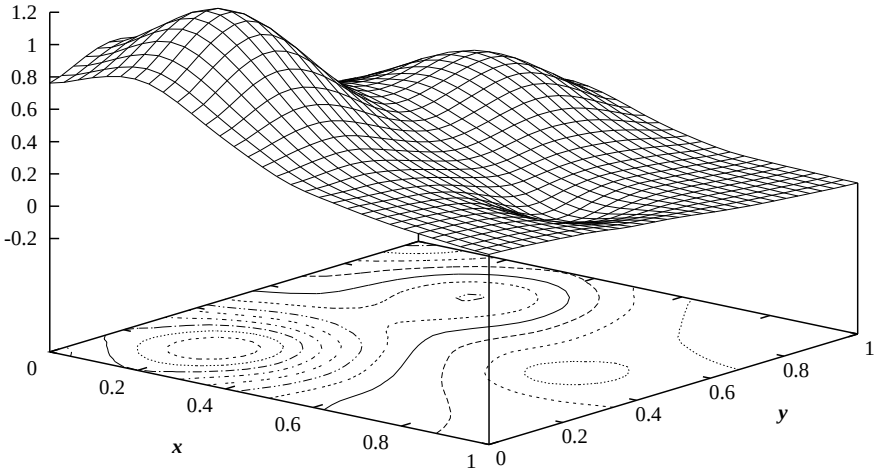
## 9.2 Program Data

```
E02JDF Example Program Data
 100                                          : number of data points to fit
  1                                           : no. points for vector evaluation
  0  0                                        : (x_i,y_i) vector to eval.
  101  101                                    : (n_x,n_y) size for mesh eval.
  0  0                                        : mesh lower-left corner
  1  1                                        : mesh upper-right corner
  F                                           : display the computed mesh vals?
```

## 9.3 Program Results

```
 E02JDF Example Program Results

 Computing the coefficients of a C^1 spline approximation to Franke's function
 Using a  6 by  6 grid
 Using an averaged local approximation

 Values of computed spline at (x_i,y_i):

          x_i         y_i  f(x_i,y_i)
         0.00        0.00        0.76

 Outputting of the function values on the mesh is disabled
```

**Example Program**
Calculation and Evaluation of Bivariate Spline Fit
from Scattered Data using Two-Stage Approximation




## 10 Optional Parameters

Several optional parameters in E02JDF control aspects of the algorithm, methodology used, logic or output. Their values are contained in the arrays IOPTS and OPTS; these must be initialized before calling E02JDF by first calling E02ZKF with OPTSTR set to '**`Initialize = E02JDF`**'.

Each optional parameter has an associated default value; to set any of them to a nondefault value, or to reset any of them to the default value, use E02ZKF. The current value of an optional parameter can be queried using E02ZLF.

The remainder of this section can be skipped if you wish to use the default values for all optional parameters.

The following is a list of the optional parameters available. A full description of each optional parameter is provided in Section 10.1.

Averaged Spline

Minimum Singular Value LPA

Polynomial Starting Degree

### 10.1 Description of the Optional Parameters

For each option, we give a summary line, a description of the optional parameter and details of constraints.

The summary line contains:

the keywords;

a parameter value, where the letters $a$, $i$ and $r$ denote options that take character, integer and real values respectively;

the default value.

Keywords and character values are case insensitive.

For E02JDF the maximum length of the parameter CVALUE used by E02ZLF is 6.

**Averaged Spline** $a$ Default $=$ 'NO'

When the bounding box is triangulated there are 8 equivalent configurations of the mesh. Setting **Averaged Spline** $=$ 'YES' will use the averaged value of the 8 possible local polynomial approximations over each triangle in the mesh. This usually gives better results but at (about 8 times) higher computational cost.

Constraint: **Averaged Spline** $=$ 'YES' or 'NO'.

**Minimum Singular Value LPA** $r$ Default $= 1.0$

A tolerance measure for accepting or rejecting a local polynomial approximation (LPA) as reliable.

The solution of a local least squares problem solved on each triangle subdomain is accepted as reliable if the minimum singular value $\sigma$ of the matrix (of Bernstein polynomial values) associated with the least squares problem satisfies **Minimum Singular Value LPA** $\leq \sigma$.

In general the approximation power will be reduced as **Minimum Singular Value LPA** is reduced. (A small $\sigma$ indicates that the local data has hidden redundancies which prevent it from carrying enough information for a good approximation to be made.) Setting **Minimum Singular Value LPA** very large may have the detrimental effect that only approximations of low degree are deemed reliable.

**Minimum Singular Value LPA** will have no effect if **Polynomial Starting Degree** $= 0$, and it will have little effect if the input data is 'smooth' (e.g., from a known function).

A calibration procedure (experimenting with a small subset of the data to be fitted and validating the results) may be needed to choose the most appropriate value for this parameter.

*Constraint*: **Minimum Singular Value LPA** $\geq 0.0$.

**Polynomial Starting Degree** $i$ Default $= 1$

The degree to be used in the initial step of each local polynomial approximation.

At the initial step the method will attempt to fit with local polynomials of degree **Polynomial Starting Degree**. If the approximation is deemed unreliable (according to **Minimum Singular Value LPA**), the degree will be decremented by one and a new local approximation computed, ending with a constant approximation if no other is reliable.

**Polynomial Starting Degree** is bounded from above by the maximum possible spline degree, 3.

The default value gives a good compromise between efficiency and accuracy. In general the best approximation can be obtained by setting **Polynomial Starting Degree** $= 3$.

*Constraint*: $0 \leq$ **Polynomial Starting Degree** $\leq 3$.

---