11.5 Least-Squares Data Fitting using K^{th} Order Splines with Constraints

A. Purpose

This package contains two subprograms, DSFIT and DSFITC, for fitting a *polynomial spline function* to discrete data. A *polynomial spline function* is a piecewise polynomial function having specified orders of continuity at the abscissae, called *internal knots*, at which one polynomial piece ends and another begins. Spline functions have been found to be very useful in many computational processes due to their capability of representing a wide variety of shapes in a controlled way.

DSFIT can be used either for a weighted least-squares fit or for interpolation. DSFITC adds capabilities for the user to specify constraints on the fit in the form of equality or inequality conditions on the value or derivative (of specified order), of the spline function at specified points, or the integral of the spline function over a specified interval. These constraints can be used, for example, to assure monotonicity or convexity of the fitted spline function over specified intervals. Figure 1 illustrates a monotone nondecreasing least-squares spline fit to data computed using DSFITC.

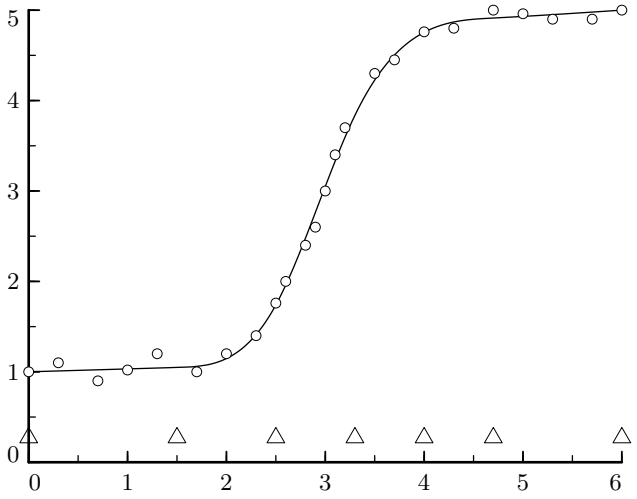

Figure 1: Constrained spline fit to data.

The fitting subprograms return coefficients of a spline function relative to *B-spline basis functions* using the parameterization conventions given by Carl de Boor in [1]. In this approach the spline function will be of a user-specified order, K, which means the polynomial pieces are of degree at most $K-1$. By default the continuity at knots will be of order K − 2, however the user can specify a lower order of continuity at selected internal knots to allow the curve to change direction more sharply.

Subprogram DSVAL can be used to evaluate a spline function represented relative to the B-spline basis or any of its derivatives, at a specified point, and subprogram DSQUAD can be used to evaluate the definite integral of the function between specified limits.

The B-spline representation has the desirable property that usual continuity conditions at knots are "built-in". This keeps down the number of coefficients that must be determined in interpolation or least-squares fitting. A disadvantage is that evaluation of a function represented in the B-spline representation is more expensive than is the use of an alternative representation using the *power* basis.

If one is going to do a large number of evaluations of a spline function one may choose to convert the representation of the spline function from the B-spline basis to the power basis to allow for more efficient evaluation. Subprogram DSTOP can be used to do this conversion. Then DPVAL can be used to evaluate the function or any of its derivatives at a specified point, and DPQUAD can be used to evaluate the definite integral of the function between specified limits.

B. Usage

Described below in Section B.1 to B.8, are:

B.1 Usage of DSFIT for fitting without constraints

B.1.a Program Prototype, Double Precision

INTEGER NXY, KORDER, NCOEF, LDW, IERR1

DOUBLE PRECISION BCOEF(≥NCOEF), **X**(≥NXY), **Y**(≥NXY), **SD**(≥NXY), **TKNOTS**(≥NCOEF+KORDER), **W**(LDW, ≥KORDER+1), **SIGFAC**

Assign values to X(), Y(), SD(), NXY, KORDER, NCOEF, TKNOTS(), and LDW.

CALL DSFIT(X, Y, SD, NXY, KORDER, NCOEF, TKNOTS, BCOEF, SIGFAC, IERR1, LDW, W)

Results are returned in BCOEF(), SIGFAC, and IERR1. Following use of DSFIT, the user may use DSVAL to compute values or specified derivatives of the fitted curve, DSQUAD to compute the definite integral of the fitted curve over a specified interval, or DSTOP to convert the representation to the power basis.

B.1.b Argument Definitions

X(), Y() [in] Data pairs (X(i), Y(i), $i = 1$, ..., NXY). Must be ordered so the X(i)'s are either nondecreasing or nonincreasing.

SD() [in] If SD(1) > 0., each SD(i) must be positive and must be the user's a priori estimate of the standard deviation of the uncertainty (*e.g.*, observational error) in the corresponding data value Y(i).

If SD(1) < 0., |SD(1)| will be used as the a priori standard deviation of each data value Y(I). In this case the array SD() may be dimensioned as SD(1).

An error condition is reported if SD(1) = 0 or if SD(1) > 0 and SD(i) ≤ 0 for $1 < i \le$ NXY.

NXY [in] Number of data points. Require NXY ≥ max(NCOEF, KORDER).

KORDER [in] Order of the spline function. Each polynomial piece will be of degree at most KORDER − 1. The default order of continuity at each internal knot will be KORDER − 2. The popular case of a cubic spline with C^2 continuity at the knots is selected by setting KORDER = 4. Require KORDER ≥ 1. Internal arrays in this package impose an upper limit of $kmax = 20$ on KORDER.

NCOEF [in] Number of terms in the sum representing the spline function. Require NCOEF ≤ NXY.

TKNOTS() [in] The knots, t_i, $i = 1$, ..., NCOEF + KORDER. The interval $[t_{\text{KORDER}}, t_{\text{NCOEF+1}}]$ will be the *proper interpolation interval* for the problem. This interval should contain all the X(j) values, so it is reasonable to set $t_{\text{KORDER}} \le \min(\text{X}(1), \text{X}(\text{NXY}))$ and $t_{\text{NCOEF+1}} \ge \max(\text{X}(1), \text{X}(\text{NXY}))$. It is convenient and reasonable to set the KORDER − 1 knots with indices less than KORDER equal to t_{KORDER}, and the KORDER − 1 knots with indices greater than NCOEF + 1 equal to $t_{\text{NCOEF+1}}$.

Knots indexed from KORDER + 1 through NCOEF are *internal* knots. Internal knots specify abscissae at which one polynomial piece ends and the next begins. Successively indexed internal knots may have the same value. A knot appearing with multiplicity μ means the order of continuity of the spline at this knot will be at least KORDER $- \mu - 1$. Require $1 \le \mu \le$ KORDER.

Require $t_i \le t_{i+1}$ for $i = 1$, ..., NCOEF + KORDER − 1; $t_i < t_{i+\text{KORDER}}$ for $i = 1$, ..., NCOEF; $t_{\text{KORDER}} < t_{\text{KORDER+1}}$; and $t_{\text{NCOEF}} < t_{\text{NCOEF+1}}$. See Sections C and D for further discussion of knot placement.

BCOEF() [out] Coefficients c_i, $i = 1$, NCOEF, in the sum representing the spline function as a sum of coefficients times B-spline basis functions.

SIGFAC [out] Set by the subroutine as a measure of the residual error of the fit. The subroutine sets

$$\text{SIGFAC} = \frac{\text{RNORM}}{\text{DOF}^{1/2}}, \quad \text{where}$$

$$\text{RNORM} = \left[\sum_{i=1}^{\text{NXY}} \left(\frac{\text{yfit}_i - \text{Y}_i}{\text{SD}_i} \right)^2 \right]^{1/2}, \quad \text{and}$$

$$\text{DOF} = \max(1, \text{NXY} - \text{NCOEF}).$$

Here SD_i denotes SD(i) if SD(1) > 0, and |SD(1)| otherwise.

IERR1 [out] Error status indicator. Set on the basis of tests done in DSFIT as well as error indicators IERR2 set by DBACC and IERR3 set by DBSOL, both in Chapter 4.5. IERR1 is set as follows:

IERR1	**Meaning**
0	No errors detected.
100	NCOEF < 1 or NCOEF > NXY
150	KORDER > $kmax$ (= 20)
200	TKNOTS(i) > TKNOTS(i+1)
250	TKNOTS(i) ≥TKNOTS(i+KORDER)
300	LDW < NCOEF + 2
400	The X(i)'s are neither nondecreasing nor nonincreasing.
600	LDW < NCOEF + 2.
700+IERR2	DBACC set IERR2 ≠ 0
800+IERR2	DBACC set IERR2 ≠ 0
900+IERR2	DBACC set IERR2 ≠ 0
1000+IERR3	DBSOL set IERR3 ≠ 0. Indicates singularity.
1100	SD(1) = 0.0
1200	SD(1) > 0.0, and SD(i) ≤ 0.0 for some $i \in$ [2, NXY].

LDW [in] Leading dimension for the work array W(). Require LDW $\geq$ NCOEF + 2. Let α denote the maximum number of data abscissae, X(i), in any one knot interval, *i.e.* between TKNOTS(j) and TKNOTS($j+1$) for some j. The subroutine will be more efficient if LDW is at least NCOEF $+ 1 + \alpha$.

W(,) [scratch] Working space, dimensioned W(LDW, $\geq$KORDER+1).

B.2 Usage of DSFITC for fitting with constraints

B.2.a Program Prototype, Double Precision

INTEGER KORDER, NCOEF, ISET(3), **INFO**(*ninfo*)

DOUBLE PRECISION X(*mdim*), **Y**(*mdim*), **SD**(*mdim*), **TKNOTS**($\geq$NCOEF+KORDER), **BCOEF**($\geq$NCOEF), **RNORM, W**(*nwork*)

CHARACTER*4 CCODE(*mdim*)

The dimension *mdim* must be large enough to provide for specification of all constraint and least-squares equations as described below in the description of CCODE(). See ISET() for the specifications of *ninfo* and *nwork*.

Assign values to CCODE(), X(), Y(), SD(), KORDER, NCOEF, TKNOTS(), and ISET().

CALL DSFITC(CCODE, X, Y, SD, KORDER, NCOEF, TKNOTS, BCOEF, RNORM, ISET, INFO, W)

Computed quantities are returned in BCOEF(), RNORM, and INFO(). Following use of DSFITC, the user may use DSVAL to compute values or specified derivatives of the fitted curve, DSQUAD to compute the definite integral of the fitted curve over a specified interval, or DSTOP to convert the representation to the power basis.

B.2.b Argument Definitions

CCODE() [in] CCODE(i), or in some cases CCODE(i) and CCODE($i+1$) together, give specifications for one constraint equation or one least-squares equation. CCODE(i) is regarded as consisting of four single-character fields.

CCODE(i)(1:1) = $kind_i$ = '1', '2', '3', '4'.
CCODE(i)(2:2) = $deriv_i$ = '0', '1', ..., '9'.
CCODE(i)(3:3) = $relop_i$ = '$\sim$', '=', '<', '>'.
CCODE(i)(4:4) = $active_i$ = 'A', 'N', '!'.

Where alphabetic characters are shown, the corresponding lower case character is also acceptable.

$active_i$ = '!' signals the end of information in this array. The user must provide this termination signal. The other fields in this array element will be ignored. $active_i$ = 'A' means CCODE(i) is active so CCODE(i) will be processed. $active_i$ = 'N' means CCODE(i) is inactive so processing will advance to CCODE(i+1). To activate or inactivate a pair [CCODE(i), CCODE($i+1$)] in which $kind_i = 3$ or 4, place the same code 'A' or 'N' in both $active_i$ and $active_{i+1}$.

$relop_i$ = '=', '<', or '>' denotes a constraint equation with '=' meaning equal to, '<' meaning less than or equal to, and '>' meaning greater than or equal to. $relop_i$ = '$\sim$' denotes a least-squares equation. For a least-squares equation the value SD(i) (or |SD(1)|) will be used as the a priori standard deviation.

$kind_i = 1$ specifies an equation of the form

$$f^{(di)}(\mathrm{X}(i))\ relop_i\ \mathrm{Y}(i)$$

where $f^{(di)}$ denotes the derivative of order $deriv_i$ of the spline function to be determined. The zeroth order derivative is the function itself.

$kind_i = 2$ specifies an equation of the form

$$f^{(di)}(\mathrm{X}(i)) - f^{(di)}(\mathrm{Y}(i))\ relop_i 0$$

Note that Y(i) is an independent variable value in this case.

$kind_i = 3$ uses items indexed by both i and $i+1$ and specifies an equation of the form

$$f^{(di)}(\mathrm{X}(i)) - \mathrm{Y}(i+1) \times f^{(di+1)}(\mathrm{X}(i+1))\ relop_i \mathrm{Y}(i)$$

where $f^{(di+1)}$ denotes the derivative of order $deriv_{i+1}$ of the spline function to be determined.

$kind_i = 4$ uses items indexed by both i and $i+1$ and specifies an equation of the form

$$\int_{\mathrm{X}(i)}^{\mathrm{X}(i+1)} f(x)\,dx \quad relop_i\ \mathrm{Y}(i)$$

See Section C for discussion of expected applications of these different equation forms.

X(), Y() [in] Data for use in building constraint or fitting equations as specified by the contents of CCODE().

SD() [in] SD(i) specifies the a priori standard deviation of the error in the equation specified by CCODE(i) when $relop_i$ = '$\sim$'. The weighted fitting algorithm will take account of these SD(i) values. Optionally, the user may set SD(1) to a negative value. Then this subroutine will use |SD(1)| as the standard deviation for the right-side value in each fitting equation. In this latter case the SD() array can be dimensioned

SD(1). Note that a negative value in SD(1) will always be interpreted in this way regardless of the contents of CCODE(1). An error condition is reported if SD(1) = 0 or if SD(1) > 0 and SD(i) $\leq$ 0 for $1 < i \leq$ NXY.

KORDER [in] Order of the spline function. Each polynomial piece will be of degree at most KORDER − 1. The default order of continuity at each internal knot will be KORDER − 2. The popular case of a cubic spline with C^2 continuity at the knots is selected by setting KORDER = 4. Require KORDER $\geq$ 1. Internal arrays in this package impose an upper limit of $kmax = 20$ on KORDER.

NCOEF [in] Number of terms in the sum representing the spline function.

TKNOTS() [in] The knots, t_i, i = 1, ..., NCOEF + KORDER. The interval $[t_{\text{KORDER}}, t_{\text{NCOEF+1}}]$ will be the *proper interpolation interval* for the problem. This interval should contain all the abscissa values occurring in the least-squares and constraint equations, so it is reasonable to set t_{KORDER} less than or equal to the minimum of these abscissae and $t_{\text{NCOEF+1}}$ greater than or equal to the maximum. It is convenient and reasonable to set the KORDER − 1 knots with indices less than KORDER equal to t_{KORDER}, and the KORDER − 1 knots with indices greater than NCOEF + 1 equal to $t_{\text{NCOEF+1}}$.

Knots indexed from KORDER + 1 through NCOEF are *internal* knots. Internal knots specify abscissae at which one polynomial piece ends and the next begins. Successively indexed internal knots may have the same value. A knot appearing with multiplicity μ means the order of continuity of the spline at this knot will be at least KORDER $- \mu - 1$. Require $1 \leq \mu \leq$ KORDER.

Require $t_i \leq t_{i+1}$ for i = 1, ..., NCOEF + KORDER − 1; $t_i < t_{i+\text{KORDER}}$ for i = 1, ..., NCOEF; $t_{\text{KORDER}} < t_{\text{KORDER+1}}$; and $t_{\text{NCOEF}} < t_{\text{NCOEF+1}}$. See Sections C and D for further discussion of knot placement.

BCOEF() [out] Coefficients c_i, i = 1, NCOEF, in the sum representing the spline function as a sum of coefficients times B-spline basis functions.

RNORM [out] Set by the subroutine as a measure of the residual error of the fit. RNORM= $\left[\sum_i \left(\frac{resid_i}{\text{SD}_i}\right)^2\right]^{1/2}$, where the summation is over indices for which $relop_i = '\sim'$, and $resid_i$ denotes the residual after the fit in the equation specified by CCODE(i). Here SD$_i$ denotes SD(i) if SD(1) > 0, and |SD(1)| otherwise.

ISET() [in] Array of length 3. These specifications use the following values:

$ns =$ the number of elements in CCODE() containing $relop = '<'$ or $'>'$ and $active = 'A'$.

$m1 =$ the number of elements in CCODE() containing $relop = '='$, $'<'$ or $'>'$ and $active = 'A'$.

$mfit =$ the number of elements in CCODE() containing $relop = '\sim'$ and $active = 'A'$.

$ntot =$ NCOEF + ns,

$mtot = m1 + mfit$,

$minmn = \min(mtot, ntot)$.

ISET(1) = *ninfo*, the dimension of INFO(). A sufficiently large value is $7 + 2\,ntot$.

ISET(2) = *nwork*, the dimension of WORK(). A sufficiently large value is $nwork = mtot \times ntot + 3\,mtot + 6\,ntot + 3\,minmn + m1$.

ISET(3) = *kprint*, a diagnostic print flag in the range [0, 4]. It is passed on to DBLSE. Zero means no printing. Larger values produce more printing.

INFO() [out and scratch] The first 7 elements of INFO() are used to return information about the problem. The following $2 \times$ (NCOEF + ns) locations are used as scratch. The dimension of INFO() is *ninfo* given in ISET(1).

INFO(1) = *ierr*5, a status indicator incorporating information from IERR4 issued by DBLSE. Possible values of *ierr5* are as follows:

- 0 No errors detected.
- 100 NCOEF < 1
- 150 KORDER > $kmax$(= 20)
- 200 TKNOTS(i) > TKNOTS(i + 1)
- 250 TKNOTS(i) $\geq$ TKNOTS(i + KORDER)
- 300 *ninfo* or *nwork* is too small. Recommended values are returned in INFO(2) and INFO(3).
- 500 $deriv_i$ has bad value for some i.
- 600 $relop_i$ has bad value for some i.
- 700 $kind_i$ has bad value for some i.
- 800 $active_i$ has bad value for some i.
- 1000 + IERR4 IERR4 $\neq$ 0 due to error detected in DBLSE. For the interpretation of IERR4 see Section E.
- 1100 SD(1) = 0.0
- 1200 SD(1) > 0.0, and SD(i) $\leq$ 0.0 for some i for which $relop_i = '\sim'$.

INFO(2) = *need*1, the dimension needed for INFO().

INFO(3) = *need*2, the dimension needed for WORK().

INFO(4) = $m1$, the number of constraint rows in the matrix representation of the problem. This will be the number of elements in CCODE() containing $relop = '=', '<'$ or $'>'$ and $active = 'A'$

INFO(5) = $mfit$, the number of least-squares equations. This will be the number of elements in CCODE() containing $relop = '\sim'$ and $active = 'A'$.

INFO(6) = ns, the number of slack variables. This will be the number of elements in CCODE() containing $relop = '<'$ or $'>'$ and $active = 'A'$.

INFO(7) = $nsets$, the number of variables in "Set S" at termination. These variables are at values determined by solution of a system of equations. The other NCOEF + ns − $nsets$ variables will be at fixed values, either at one of their bounds or at zero.

WORK() [scratch] Work space dimensioned *nwork*. See ISET(2) above.

B.3 Usage of DSVAL for evaluation using the B-spline basis

DSVAL returns the value at argument X of the derivative of order IDERIV of the spline function defined by the parameter sequence [KORDER, NCOEF, TKNOTS(), BCOEF()].

B.3.a Program Prototype, Double Precision

INTEGER NCOEF, KORDER, IDERIV

DOUBLE PRECISION DSVAL,
TKNOTS(≥NCOEF+KORDER),
BCOEF(≥NCOEF), **X**

Assign values to all arguments.

D = DSVAL(KORDER, NCOEF, TKNOTS, BCOEF, X, IDERIV)

B.3.b Argument Definitions

KORDER, NCOEF, TKNOTS(), BCOEF() [in] Quantities defining a spline function relative to the B-spline basis as returned by DSFIT or DSFITC. Internal arrays in this subprogram impose an upper limit of $kmax = 20$ on KORDER.

X [in] Argument at which the IDERIV order derivative of the spline function will be evaluated.

IDERIV [in] Derivative order desired. Require IDERIV ≥ 0. Zero means to evaluate the spline function itself.

B.4 Usage of DSQUAD for integration using the B-spline basis

DSQUAD returns the value of the integral from X1 to X2 of the spline function defined by the parameter sequence [KORDER, NCOEF, TKNOTS(), BCOEF()].

B.4.a Program Prototype, Double Precision

INTEGER KORDER, NCOEF

DOUBLE PRECISION DSQUAD,
TKNOTS(≥NCOEF+KORDER),
BCOEF(≥NCOEF), **X1, X2**

Assign values to all arguments.

D = DSQUAD(KORDER, NCOEF, TKNOTS, BCOEF, X1, X2)

B.4.b Argument Definitions

KORDER, NCOEF, TKNOTS(), BCOEF() [in] Quantities defining a spline function relative to the B-spline basis as returned by DSFIT or DSFITC.

X1, X2 [in] Limits of the integral to be evaluated. Permit X1 < X2 or X1 ≥ X2.

B.5 Usage of DSTOP to convert from the B-spline basis to the power basis

DSTOP converts the representation of a spline function from the B-spline parameterization [KORDER, NCOEF, TKNOTS(), BCOEF()] to the power basis form [KORDER, NPC, XI(), PCOEF()]. KORDER will not be changed. Typically the B-spline parameters will have come from DSFIT or DSFITC. The power coefficients can be used by DPVAL and DPQUAD.

B.5.a Program Prototype, Double Precision

INTEGER KORDER, NCOEF, NPC

DOUBLE PRECISION TKNOTS(≥NCOEF+KORDER), **BCOEF**(≥NCOEF),
BDIF(≥ NCOEF×KORDER), **XI**(*mpc*+1),
PCOEF(≥KORDER×*mpc*)

The dimension *mpc* must be as large as the output value NPC. In terms of input quantities it suffices to set *mpc* ≥ NCOEF − KORDER + 1. Assign values to KORDER, NCOEF, TKNOTS(), and BCOEF().

CALL DSTOP(KORDER, NCOEF, TKNOTS, BCOEF, BDIF, NPC, XI, PCOEF)

Results are returned in NPC, XI(), and PCOEF().

B.5.b Argument Definitions

KORDER, NCOEF, TKNOTS(), BCOEF() [in] Quantities defining a spline function relative to the B-spline basis, as returned by DSFIT or DSFITC.

BDIF() [scratch] Work space of size NCOEF × KORDER.

NPC [out] NPC + 1 will be the number of distinct values in the sequence [TKNOTS(i), i = KORDER, ..., NCOEF + 1]. NPC will satisfy NPC ≤ NCOEF − KORDER + 1.

XI() [out] A strictly increasing sequence of length NPC + 1 consisting of all the distinct values from the sequence [TKNOTS(i), i = KORDER, ..., NCOEF + 1].

PCOEF() [out] PCOEF($i + (j-1)\times$ KORDER) will be be the coefficient of $(t - \text{XI}(j))^{(i-1)}$ in the power basis representation of the spline function. (i = 1, ..., KORDER; j = 1, ..., NPC)

B.6 Usage of DPVAL for evaluation using the power basis

DPVAL returns the value at argument X of the derivative of order IDERIV of the spline function defined by the parameter sequence [KORDER, NPC, XI(), PCOEF()].

B.6.a Program Prototype, Double Precision

INTEGER NPC, KORDER, IDERIV

DOUBLE PRECISION DPVAL,
XI(≥NPC+KORDER),
PCOEF(≥KORDER×NPC), **X**

Assign values to all arguments.

D = DPVAL(KORDER, NPC, XI, PCOEF, X, IDERIV)

B.6.b Argument Definitions

KORDER, NPC, XI(), PCOEF() [in] Quantities defining a spline function relative to the power basis, as returned by DSTOP.

X [in] Argument at which the IDERIV order derivative of the spline function will be evaluated.

IDERIV [in] Derivative order desired. Require IDERIV ≥ 0. Zero means to evaluate the spline function itself.

B.7 Usage of DPQUAD for integration using the power basis

DPQUAD returns the value of the integral from X1 to X2 of the spline function defined by the parameter sequence [KORDER, NPC, XI(), PCOEF()].

B.7.a Program Prototype, Double Precision

INTEGER KORDER, NPC

DOUBLE PRECISION DPQUAD,
XI(≥NPC+KORDER),
PCOEF(≥KORDER×NPC), **X1, X2**

Assign values to all arguments.

D = DPQUAD(KORDER, NPC, XI, PCOEF, X1, X2)

B.7.b Argument Definitions

KORDER, NPC, XI(), PCOEF() [in] Quantities defining a spline function relative to the power basis, as returned by DSTOP.

X1, X2 [in] Limits of the integral to be evaluated. Permit X1 < X2 or X1 ≥ X2.

B.8 Modifications for Single Precision

For single precision usage change the DOUBLE PRECISION statements to REAL and change the initial "D" in the subprogram names to "S".

C. Examples and Remarks

C.1 Demonstration of DSFIT.

The demonstration driver DRDSFIT sets up a curve fitting problem having 12 (x, y) pairs of data. It uses DSFIT to do a least-squares fit to this data with an 8-parameter cubic spline function. It uses DSVAL to evaluate the fitted function over the given set of x values. It uses DSQUAD to compute the definite integral of the fitted function from 5.0 to 20.0. It uses DSTOP to convert the B-spline representation to the power representation. It then uses DPVAL and DPQUAD to repeat the function evaluation and integral computation using the power representation. The output is listed in ODDSFIT.

C.2 Demonstration of DSFITC.

The program DRDSFITC illustrates the use of DSFITC to compute a constrained least-squares spline fit to data. Output from DRDSFITC is listed in ODDSFITC and graphs prepared using `splot` from Chapter 16-03 are shown in Figures 1 and 2. We have 24 data points, given in the the first 24 entries of the arrays XI() and YI() in the DATA statement in DRDSFITC. These points are shown as circles in Figure 1.

Suppose these data are measurements of some phenomenon that is known to be monotone nondecreasing and we wish to find a monotone nondecreasing function that closely fits the data. An unconstrained least-squares fit to this data by a single polynomial or by a polynomial spline function will have unwanted oscillations.

There are rational functions and exponential functions with three parameters that are monotone and of somewhat the desired shape, but these functions do not have enough free parameters to allow the function to fit the data really closely. A satisfactory fit can be obtained using a cubic spline function having C^2 continuity.

The data abscissae range from 0 to 6. We shall place quadruple knots at these two points and internal knots at 1.5, 2.5, 3.3, 4.0, and 4.7. The number and locations of these internal knots were selected by some trial and error. These knot values are stored in TKNOTS(). Their locations are shown by triangles in Figures 1 and 2. Since we have selected a total of 13 knots and we have KORDER = 4 to specify a cubic spline, the number of coefficients will be set to NCOEF = 13 − 4 = 9.

We shall require the curve to be concave up over [0, 2.5] by requiring $f'' \geq 0$ at 0 and at the first two internal knots. These constraints, along with the constraint $f'(0) \geq 0$, will forcef' to be nonnegative over [0, 2.5]. Similarly we require the curve to be concave down over [3.3, 6] by requiring $f'' \leq 0$ at the last three internal knots and at 6. These constraints, along with the constraint $f'(6) \geq 0$, will force f' to be nonnegative over [3.3, 6]. It follows that $f' \geq 0$ over [2.5, 3.3] since in this interval f' is a quadratic polynomial that is nonnegative and nondecreasing at 2.5 and nonnegative and nonincreasing at 3.3.

The second derivative of a cubic spline is linear between knots. Thus the conditions $f''(2.5) \geq 0$ and $f''(3.3) \leq 0$ imply that f'' can have at most one sign change in the interval between the successive knots at 2.5 and 3.3. This assures the only inflection point of the curve over (0, 6) will occur in the interval [2.5, 3.3]. See Figure 2 for graphs of the resulting f' and f''.

Supposing we also wish to have the fitted curve take the value 1 at 0 and the value 5 at 6, we also impose these constraints.

Recall that the four characters in each entry of CCODE() are interpreted as (*kind*, *deriv*, *relop*, *active*). All but the last element of CCODE() have *active* = 'a' meaning these elements are active, while the last element has *active* = '!', which is the termination signal. All of the active elements have *kind* = '1', meaning the specified equation is of the form

$$f^{(di)}(\mathrm{XI}(i))\ relop_i\ \mathrm{YI}(i).$$

The first 24 entries have *relop* = '∼' meaning that each specifies one of the least-squares equations. The ten elements of CCODE() beginning with CCODE(25) specify the constraints. For example CCODE(26) has *deriv* = '1' and *relop* = '>' meaning the first derivative at XI(26) is constrained to be ≥ YI(26).

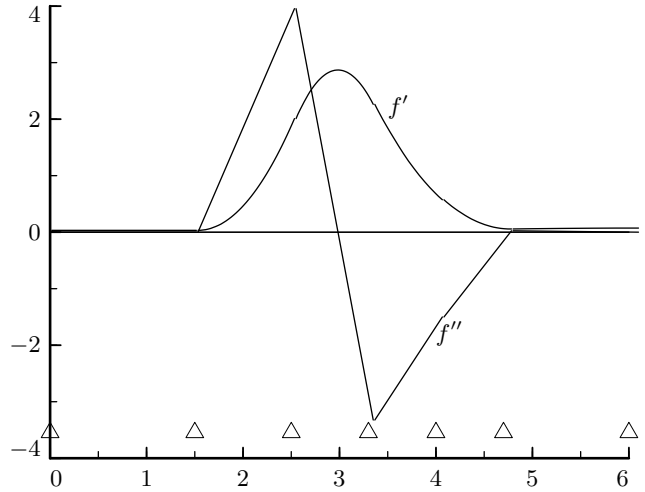

Figure 2: First and second derivatives of fitted function.

C.3 Using constraints to control shape.

A function with at least C^1 continuity is nondecreasing over [c, d] if its first derivative is nonnegative throughout [c, d]. A function with at least C^2 continuity is concave up over an interval [c, d] if its second derivative is nonnegative throughout [c, d]. Although these are properties defined over an interval, it is possible to impose these conditions on spline functions by appropriate assignment of constraints at a finite number of points.

Note that if f is a cubic spline function with C^2 continuity, then f' is a quadratic spline with C^1 continuity, and f'' is a linear spline function with C^0 continuity, *i.e.*, f'' is a continuous piecewise linear function with possible slope changes only at knots. It follows that by requiring $f'' \geq 0$ at c and d, and at any knots between c and d, f'' will necessarily be nonnegative throughout [c, d], and therefore f will be concave up throughout [c, d].

If one wants f to be monotone nondecreasing as well as concave up over [c, d] it suffices to require $f'(c) \geq 0$ along with the second derivative conditions already discussed, since with $f'' \geq 0$ throughout [c, d], f' cannot have a smaller value anywhere in [c, d] than it has at c.

If one wants monotonicity for f without second derivative constraints, one could let f be a quadratic spline rather than a cubic spline. Then f' will be piecewise linear and one can control the sign of f' over an interval by constraining f' at knots as was done above for f''. If one prefers to use cubic splines, one can do trial and error placement of constraints on f' and eventually keep f' from changing sign.

C.4 Periodicity.

When periodicity is desired it should be specified for the function value and all orders of derivatives that are continuous at the points referenced in the specification. For

example suppose one wants periodicity of 360 degrees. The *proper interpolation interval* could be set as [a, b] = [0.0, 360.0]. If one uses KORDER = 4 one should specify periodicity for f, f', and f''. This can be done by setting CCODE(1:3) = ′20=a′, ′21=a′, ′22=a′; X(1:3) = 0.0, 0.0, 0.0; and Y(1:3) = 360.0, 360.0, 360.0.

Although it does not change the fit obtained, one may wish to have the periodicity reflected in the coefficients. Letting p denote the period, and assuming $p + t_{\mathrm{KORDER}} = p + a = b = t_{\mathrm{NCOEF}+1}$, this can be done by setting the initial knots as $t_i = -p + t_{\mathrm{NCOEF}+1-\mathrm{KORDER}+i}$, for $i = 1$, ..., KORDER − 1, and the final knots as $t_{\mathrm{NCOEF}+1+i} = p + t_{\mathrm{KORDER}+i}$, for $i =$ 1, ..., KORDER − 1. Then the coefficients will reflect the periodicity by satisfying $c_{\mathrm{NCOEF}+1-\mathrm{KORDER}+i} = c_i$, for $i = 1$, ..., KORDER − 1.

C.5 Differential equations.

Using $kind = 3$, conditions such as $f'(x) - cf(x) = d$, or $f'(x) - cf(x) \sim d$ (and slightly more general expressions) can be specified for given values of x, c and d. Thus DSFITC can be used to implement the collocation method of computing an approximate solution to linear differential equations.

C.6 Assignment of knots.

Many discussions of spline interpolation are based on the assumption that many, or all, of the knots will be assigned to coincide with data abscissae. This is not necessary, either for interpolation or least-squares fitting.

Let, $\{B_i\colon\ i = 1, ..., \mathrm{NCOEF}\}$, be a family of B-spline basis functions of order K. An unconstrained interpolation or least-squares fitting problem using this family gives rise to a full-rank matrix, and thus has a unique solution if and only if there are at least NCOEF distinct data abscissae, and it is possible to choose NCOEF of these and relabel them, say as u_i, $i = 1$, NCOEF, so they will satisfy $B_i(u_i) \neq 0$ for $i = 1$, ..., NCOEF. This condition will be satisfied if the (possibly relabled) u_i's relate to the knots according to $t_i < u_i < t_{i+K}$ for $i = 1$, ..., NCOEF.

Consider an interpolation problem with NXY data points, all data abscissae being distinct. Choose a spline order $K \geq 2$. We must use exactly NCOEF = NXY B-spline basis functions. Thus NCOEF + K knots must be assigned. Let a and b be the minimum and maximum data abscissae respectively. Assign the first K knots the value a and the last K knots the value b. Then NCOEF − K knots remain to be assigned and there are NCOEF − 2 data abscissae distinct from a and b. One simple approach is to use any NCOEF − K of these NCOEF − 2 data abscissae as knots. In the popular case of cubic spline interpolation($K = 4$) there would be just two data abscissae not used as knots. It is common to choose these to be the first one after a and the last one before b.

Another method suggested in [1], pp. 218–219, for assigning the interior knots for interpolation is the formula

$$t_i = (u_{i-K+1} + ... + u_{i-1})/(K-1), \qquad i = K+1,\ ...,\ NCOEF$$

where the ordered set of data abscissae is denoted by $\{u_i\colon\ i = 1, ..., \mathrm{NCOEF}\}$.

For least-squares fitting one must choose NCOEF < NXY.

D. Functional Description

D.1 Representation of an individual B-spline basis function.

Let $\{t_1, ..., t_{K+1}\}$ be a sequence of strictly increasing real numbers that we will call *knots*. To within a multiplicative scale factor, there is one, and only one, spline polynomial function of order K (*i.e.*, having polynomial pieces of degree at most $K - 1$), having at least C^{K-2} continuity at these knots, and being nonzero throughout the open interval (t_1, t_{K+1}), and zero outside this interval. With some convention for assigning the scale factor, such a function is called a B-spline basis function. The interval $\langle t_1, t_{K+1}\rangle$ will be called the *support interval* for this B-spline basis function. We use angle brackets $\langle\ \rangle$ to indicate that we are not specifying whether the endpoints are included or not. Graphs of B-spline basis functions of orders 1, 2, 3, and 4 over uniformly spaced knots are given in Figures 3–5.

Following [1], these definitions can be generalized to allow knots to coalesce. This has a natural mathematical interpretation of reducing the order of continuity at the affected knots. At an ordinary non-multiple knot, a spline function of order K has C^{K-2} continuity. At a knot of multiplicity μ a spline function of order K has $C^{K-\mu-1}$ continuity. For example a B-spline basis function of order 4 defined over the knot set {0, 1, 1, 5, 6} consists of only three nontrivial cubic pieces, and these have C^2 continuity at 0, 5, and 6, but only C^1 continuity at 1. This freedom to lower the order of continuity at specified places can be useful in allowing a curve to change direction more sharply at such a point. Graphs of B-spline basis functions of order 4 having multiple knots at the left end are given in Figure 6.

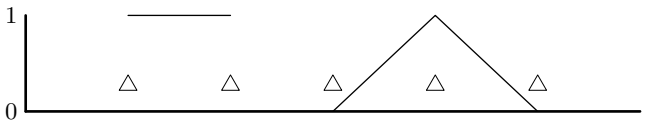

Figure 3: B-spline basis functions of orders 1 and 2.

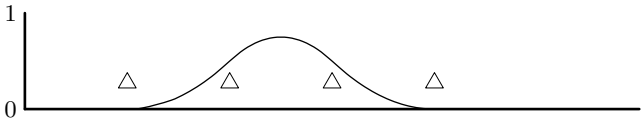

Figure 4: B-spline basis function of order 3.

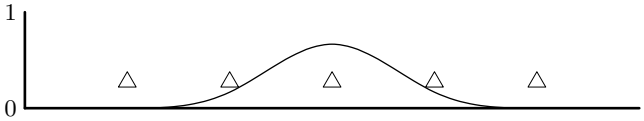

Figure 5: B-spline basis function of order 4.

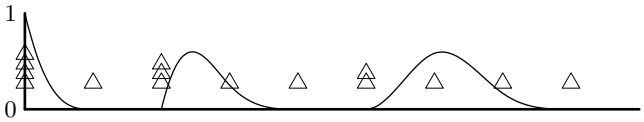

Figure 6: Cubic B-spline basis functions having multiple knots at the left.

D.2 Representation of a spline function using the B-spline basis.

Suppose we wish to construct a family of spline functions of order K, and having NCOEF degrees of freedom, over an interval $[a,\ b]$, which will be called the *proper interpolation interval* for this spline family. We require $a < b$ and NCOEF $\geq K$. Construct a *knot sequence* T $= \{t_1, ..., t_{\text{NCOEF}+K}\}$. This sequence must be nondecreasing and have $t_K = a$ and $t_{\text{NCOEF}+1} = b$. The values of the knots indexed before K or after NCOEF + 1 do not affect the shapes that can be achieved by the family of splines to be defined. A convenient way to set these knots is to set $t_1 = ... = t_K = a$, and $t_{\text{NCOEF}+1} = ... = t_{\text{NCOEF}+K} = b$.

The knots indexed from $K+1$ through NCOEF are called *internal* knots. Internal knots define where the different polynomial pieces meet. Their placement determines the shapes that the resulting spline family can achieve. We require $t_K < t_{K+1}$, and $t_{\text{NCOEF}} < t_{\text{NCOEF}+1}$, and $t_i < t_{i+K}$ for $i = 1, ...,$ NCOEF. Within the limitations of these constraints, successive interior knots need not be distinct. At an internal knot of multiplicity μ members of this spline family will have $C^{K-\mu-1}$ continuity.

For each $i = 1, ...,$ NCOEF, we define a B-spline basis function B_i having $\langle t_i,\ t_{i+K}\rangle$ as its support interval. For $i = K, ...,$ NCOEF, each interval $\langle t_i,\ t_{i+1}\rangle$ is in the support interval of exactly K basis functions, namely $\{B_j$: $j = i - K + 1, ..., i\}$. If such an interval $\langle t_i,\ t_{i+1}\rangle$ has nonzero length, the K basis functions that contain this interval in their support intervals form a basis for the space of all polynomials of degree $\leq K - 1$ over this interval. The closed union of these intervals is the interval $[a,\ b]$. This is the interval over which it is most reasonable to use linear combinations of the B_i's to fit data.

Any polynomial of degree $\leq K - 1$ can be exactly represented over $[a,\ b]$ by a linear combination of the B_i's, $i = 1, ...,$ NCOEF. In particular the constant function whose value is one is representable over $[a,\ b]$ by a linear combination of the B_i's.

In this package the scaling of the B_i's is determined by the requirement that all the coefficients in this linear combination be ones, *i.e.*, $\sum_{i=1}^{\text{NCOEF}} B_i(t) = 1$ for all $t \in [a,\ b]$.

Given coefficients, c_i, $i = 1, ...,$ NCOEF, a spline function, $f(t)$, is represented for $t \in [a,\ b]$ as $f(t) = \sum_{i=1}^{\text{NCOEF}} c_i B_i(t)$.

Although this is a sum of NCOEF terms, at most K of the terms are nonzero at any single point, t, due to the properties of the basis functions. For evaluation of $f(t)$ at a point $t \in (a, b)$ that coincides with a knot, this package uses the polynomial piece defined over the nonzero subinterval immediately to the right of t. This package allows extrapolation outside the interval (a, b) using the convention that for $t \leq a$ the package will extend the polynomial that is defined over $\langle t_K, t_{K+1}\rangle$, and for $t \geq b$ the package will extend the polynomial that is defined over $\langle t_{\text{NCOEF}}, t_{\text{NCOEF}+1}\rangle$.

Within this package a spline function is fully specified relative to the B-spline basis by two integers, KORDER and NCOEF, and two floating point arrays, TKNOTS() and BCOEF(), containing [t_i: $i = 1, ...,$ NCOEF + KORDER] and [c_i: $i = 1, ...,$ NCOEF].

D.3 Representation of a spline function using the piecewise power basis.

Assume a spline function $f(t)$ has been defined relative to the B-spline basis as described above. Let NPC be the number of subintervals of nonzero length into which $[a,\ b]$ is partitioned by the knot sequence T. Let x_j, $j = 1, ...,$ NPC be the left endpoints of these subintervals, and let $x_{NPC+1} = b$. For the half-open subinterval $[x_j, x_{j+1})$ coefficients $p_{i,j}$ can be determined so the polynomial

$$p_{1,j} + p_{2,j}h + p_{3,j}h^2 + ... + p_{K,j}h^{K-1}$$

with $h = (t - x_j)$, is identical to the polynomial spline function $f(t)$ over this interval. If evaluation for t outside $[a, b)$ is requested the package will use the coefficients indexed by $j = 1$ if $t < a$, and will use $j =$ NPC if $t \geq b$.

Within this package a piecewise polynomial represented relative to the power basis is specified by two integers,

KORDER and NPC, and two floating point arrays, XI() and PCOEF(), containing $\{x_i,\ i = 1,\ ...,\ \text{NPC} +1\}$ and $\{p_{i,j},\ i = 1,\ ...,\ \text{KORDER};\ j = 1,\ ...,\ \text{NPC}\}$.

The power representation does not inherently assure any particular order of continuity at the knots, however if the coefficients are determined by conversion from a B-spline representation they will represent the same spline function and thus have the same continuity properties.

D.4 Computation using B-spline basis functions

Suppose a spline function f is defined relative to a B-spline basis by the quantities KORDER, NCOEF, T() and BCOEF() as discussed above. The *proper interpolation interval* for f is $[a,\ b]$ where $a = t_{\text{KORDER}}$, and $b = t_{\text{NCOEF}+1}$. With any argument x we associate a *reference index,* j, and *reference interval* $\langle t_j,\ t_{j+1}\rangle$ having $t_j < t_{j+1}$. If $x \in [a, b)$, j is chosen so that $x \in [t_j, t_{j+1})$, otherwise, if $x < a$ set $j =$ KORDER, and if $x \geq b$ set $j =$ NCOEF. Given x, subprogram DSFIND determines its reference index. From an initial trial value for j the subprogram searches forward or backward, doubling the index increment for each trial, until either a bracketing pair of knots is found or the search reaches one end of the specified search range. If a bracketing interval is found bisection is used, if necessary, to reduce the interval to the prescribed form.

To describe the computational algorithms we need to consider families of lower order basis functions over the same knot sequence T. Let NT denote the number of knots in T, *i.e.*, NT = KORDER + NCOEF. For $k = 1$, ..., KORDER, let $\{B_{i,k},\ i = 1,\ ...,\ \text{NT} - k\}$, be the set of B-spline basis functions of order k associated with T. The support interval for the function $B_{i,k}$ is $\langle t_i,\ t_{i+k}\rangle$. Formally one may follow [1], p. 118, and define $B_{i,k} \equiv 0$ if $t_i = t_{i+k}$, however it happens that these functions do not occur in the algorithms we consider.

A B-spline basis function of order k can be expressed in terms of two basis functions of order $k - 1$ as

$$B_{i,k}(x) = \frac{x - t_i}{t_{i+k-1} - t_i} B_{i,k-1}(x) + \frac{t_{i+1} - x}{t_{i+k} - t_{i+1}} B_{i+1,k-1}(x). \quad (1)$$

This formula was discovered and published independently by M. G. Cox and C. de Boor in 1972. It is a very favorable formula with regard to propagation of round-off error since, except when used for extrapolation, the B's and the factors multiplying the B's are always nonnegative, so the central "+" always involves addition of nonnegative quantities.

Consider now the problem in which we are given an x and its reference index j, and we need to compute values at x of the KORDER basis functions of order KORDER that are nonzero on $\langle t_j,\ t_{j+1}\rangle$. These functions will be $B_{j-\text{KORDER}+1,\text{KORDER}}$ through $B_{j,\text{KORDER}}$.

Among the basis functions of order 1, only $B_{j,1}$ is nonzero on $\langle t_j,\ t_{j+1}\rangle$ and its value is 1 throughout this interval. At order 2 only the two basis functions $B_{j-1,2}$ and $B_{j,2}$ are nonzero on $\langle t_j,\ t_{j+1}\rangle$. These can be computed using Eq. (1) and the known values of $B_{j-1,1}$, $B_{j,1}$, and $B_{j+1,1}$, which are 0, 1, and 0, respectively. Clearly this process can be continued until the values of the KORDER nonzero basis functions of order KORDER are computed. This method is implemented in subprogram DSBASD. For the case of cubic splines (KORDER = 4), this involves nine applications of Eq. (1) and in six of these applications one of the entering B's is known to be zero.

To evaluate a spline function f at a given argument x, one could use this method to evaluate the nonzero basis functions and then form the sum of these multiplied by the coefficients that define f. There is a more efficient method however. For a spline function f of order k, its evaluation for a point x with reference index j can be expressed as $f(x) = \sum_{i=j-k+1}^{j} c_{i,k} B_{i,k}(x)$. Replacing each $B_{i,k}$ in this expression by the right side of Eq. (1) and collecting terms on the $B_{i,k-1}$'s, and noting that only $k-1$ of these $(k-1)$-order basis functions are nonzero on the reference interval, gives the expression $f(x) = \sum_{i=j-k+2}^{j} c_{i,k-1}(x) B_{i,k-1}(x)$ where

$$c_{i,k-1}(x) = \frac{(x - t_i)\, c_{i,k} + (t_{i+k-1} - x)\, c_{i-1,k}}{t_{i+k-1} - t_i}. \quad (2)$$

One can continue reducing the spline order and the number of terms in the sum in this way, finally reaching spline order 1 with only one term in the sum: $f(x) = c_{j,1}(x) B_{j,1}(x) \equiv c_{j,1}(x)$, since $B_{j,1}(x) = 1$.

Thus, as an algorithm for evaluating a spline function of order KORDER at an argument x with reference index j, one initializes the process by setting $c_{i,\text{KORDER}} = c_i$, for $i = j -$ KORDER $+ 1$, ..., j. Then for $k =$ KORDER, KORDER $- 1$, ..., 2, one computes $c_{i,k-1}$ for $i = j-k+2$, ..., j, using Eq. (2). The final quantity $c_{j,1}$ is the value $f(x)$. This method is implemented in subprogram DSVAL. For the case of cubic splines (KORDER = 4), this involves six applications of Eq. (2).

The first derivative of a spline function f of order k is a spline function of order $k - 1$ over the same knot sequence. For an x with reference index j we have $f(x) = \sum_{i=j-k+1}^{j} c_{i,k} B_{i,k}(x)$ and $f'(x) = \sum_{i=j-k+2}^{j} c^{(1)}_{i,k-1} B_{i,k-1}(x)$, where it can be shown that

$$c^{(1)}_{i,k-1} = \frac{(k-1)\,(c_{i,k} - c_{i-1,k})}{t_{i+k-1} - t_i}. \quad (3)$$

To compute the value of the n^{th} derivative of a spline function, Eq. (3) can be applied as many times as necessary to compute coefficients of a B-spline representation of the n^{th} derivative, and then Eq. (2) can be used to evaluate the derivative. This algorithm is implemented in DSVAL with all computation being done from scratch for a given x.

Since Eq. (3) does not involve x, it is possible to use Eq. (3) to precompute an array of coefficients for later use in computing derivative values for many x values. This approach is implemented in DSDIF and DSVALA. DSDIF computes the array of coefficients for all derivatives of orders up to a specified NDERIV and DSVALA uses these coefficients in computing the values of all derivatives of orders up to NDERIV for a given x. Subprogram DSTOP for the conversion from the B-spline basis to the power basis uses DSVALA, since the coefficients relative to the power basis are just derivatives of the spline function divided by factorials.

For an x with reference index j, DSBASD computes the NDERIVth derivative of the KORDER basis functions of order KORDER that are nonzero on the reference interval. From Eq. (3) we can express the first derivative of a single basis function as

$$B'_{i,k} = (k-1)\left[\frac{B_{i,k-1}(x)}{t_{i+k-1}-t_i} - \frac{B_{i+1,k-1}(x)}{t_{i+k}-t_{i+1}}\right]. \qquad (4)$$

DSBASD first uses Eq. (1) to compute the values $B_{i,\mathrm{KORDER-NDERIV}}(x)$, for $i = j -$ KORDER + NDERIV + 1, ..., j. Then for $d = 1$, ..., NDERIV, DSBASD uses Eq. (4) to compute $B^{(d)}_{i,\mathrm{KORDER-NDERIV}+d}(x)$, for $i = j -$ KORDER + NDERIV $- d + 1$, ...,j.

It is possible to derive formulas for the exact integration of a spline function by appropriate inverse use of Eq. (3). The resulting method is unwieldy, and suspect with regard to propagation of round-off error. Instead we follow the approach of Amos, [2], that uses Gaussian quadrature. An n-point Gaussian quadrature formula is exact for polynomials up to degree 2n-1. The formula is applied separately to each polynomial piece needed to cover a specified integration interval. This method is used in DSQUAD and in DSBASI. Each of these subprograms contains stored constants for 2, 6, and 10-point Gaussian formulas. The 2-point formula is used for KORDER from 1 to 4, the 6-point formula from 5 to 12, and the 10-point formula from 13 to 20.

The fitting subroutine DSFIT uses DSBASD to form rows of the matrix for the least-squares problem. Each row will have at most KORDER nonzeros in consecutive locations giving rise to a block-banded form for the matrix. DSFIT uses DBACC and DBSOL (Chapter 4.5) to process and solve this system. This approach takes advantage of both the band structure and sequential processing to reduce the amount of working space needed.

The constrained fitting subroutine DSFITC uses DSBASD and DSBASI, as appropriate to form rows of matrices representing the constraint conditions and the least-squares problem. Due to the general form of constraints allowed the overall problem is not assumed to have a banded form so the matrices are formed in full. The resulting problem is linear least-squares with general linear equality and inequality constraints which is solved using the lower level subroutine DBLSE.

References

1. Carl de Boor, **A Practical Guide to Splines**, Springer Verlag, Berlin (1978) 392 pages.

2. D. E. Amos. Technical Report SAND79–1825, Sandia Laboratory, Albuquerque, NM (June 1979).

3. T. M. Lang, R. J. Hanson, and D. R. Campbell, **French Curve**. Internal Computing Memorandum 203, Jet Propulsion Laboratory, Pasadena, CA (Sept. 1968) 27 pages.

E. Error Procedures and Restrictions

DSFIT, DSFITC, and DSVAL each contain an internal dimensioning parameter $kmax = 20$. It is an error if KORDER $> kmax$ in any of these subprograms.

DSFIT handles any detected error by setting IERR1, reporting the error to the library error message processing subroutines of Chapter 19.2 with LEVEL = 0 and then returning. DSFITC handles errors similarly, setting the indicator *ierr5* in INFO(1).

The only error detected in DSVAL is KORDER $> kmax$, in which case DSVAL calls the library error processing subroutines with LEVEL = 2 which nominally causes message printing and termination of execution.

Abscissae and weights for 2-point, 6-point, and 10-point Gaussian quadrature are stored to 40 decimal digits in DSQUAD. With infinite precision abscissae and weights, these formulae would be exact for splines of KORDER up to 20. DPQUAD does not use any inexact stored constants.

The lower level subroutine DBLSE is used by DSFITC to solve the constrained least-squares problem. If it detects error conditions it sets $ierr4 \neq 0$ and DSFITC returns with INFO(1) = 1000 + $ierr4$. Possible nonzero values of *ierr4* are:

−1 Failed to triangularize the *m1* general constraint equations. The subroutine attempts to complete the

computation, omitting the constraint rows not triangularized. User should check the residuals of the constraint rows.

+1 $mtot \leq 0$ or $ntot \leq 0$.

+2 Inconsistent setting of bounds.

+3 Too many iterations needed. Nominal: $itmax = 5 \times ntot$.

For definitions of *m1*, *mtot*, and *ntot* see the specification of ISET() in Section B.2.b.

F. Supporting Information

The source language for these subroutines is ANSI Fortran 77.

DSFIT and DSFITC evolved from codes originally designed by R. J. Hanson and C. L. Lawson at JPL in 1968. The initial version of DSFITC, [3], was called "French Curve" to call attention to the flexibility it provided for shaping a fitted curve. Subprograms DSVAL, DPVAL, and DSTOP are modifications by Lawson of codes developed by C. de Boor, [1]. Subprograms DSQUAD and DPQUAD are modifications by Lawson of codes due to D. E. Amos, [2].

Entry	Required Files
DPQUAD	DERV1, DPQUAD, DSFIND, ERFIN, ERMSG, IERM1, IERV1
DPVAL	DERV1, DPVAL, DSFIND, ERFIN, ERMSG, IERM1, IERV1
DSFIT	DBACC, DBSOL, DERV1, DHTCC, DNRM2, DSBASD, DSFIT, ERFIN, ERMSG, IERM1, IERV1
DSFITC	DAXPY, DBLSE, DBSOL, DCOPY, DDOT, DHTCC, DERV1, DNRM2, DRANU, DROTG, DSBASD, DSBASI, DSFIND, DSFITC, DSWAP, ERFIN, ERMOR, ERMSG, IERM1, IERV1, RANPK1, RANPK2
DSQUAD	DERV1, DSBASD, DSFIND, DSQUAD, DSVAL, DSVALA, ERFIN, ERMSG, IERM1, IERV1
DSTOP	DERV1, DSBASD, DSDIF, DSFIND, DSTOP, DSVALA, ERFIN, ERMSG, IERM1, IERV1
DSVAL	DERV1, DSFIND, DSVAL, ERFIN, ERMSG, IERM1, IERV1
SPQUAD	ERFIN, ERMSG, IERM1, IERV1, SERV1, SPQUAD, SSFIND
SPVAL	ERFIN, ERMSG, IERM1, IERV1, SERV1, SPVAL, SSFIND
SSFIT	ERFIN, ERMSG, IERM1, IERV1, SBACC, SBSOL, SERV1, SHTCC, SNRM2, SSBASD, SSFIT
SSFITC	ERFIN, ERMOR, ERMSG, IERM1, IERV1, RANPK1, RANPK2, SAXPY, SBLSE, SBSOL, SCOPY, SDOT, SHTCC, SERV1, SNRM2, SRANU, SROTG, SSBASD, SSBASI, SSFIND, SSFITC, SSWAP
SSQUAD	ERFIN, ERMSG, IERM1, IERV1, SERV1, SSBASD, SSFIND, SSQUAD, SSVA, SSVALA
SSTOP	ERFIN, ERMSG, IERM1, IERV1, SERV1, SSBASD, SSDIF, SSFIND, SSTOP, SSVAL
SSVAL	ERFIN, ERMSG, IERM1, IERV1, SERV1, SSFIND, SSVAL

DRDSFIT

```
c      program DRDSFIT
c>> 1996-07-03 DRDSFIT Krogh Special code for C conversion.
c>> 1996-06-19 DRDSFIT Krogh Changes in formats for C conversion.
c>> 1996-05-28 DRDSFIT Krogh Added external & removed Fortran 90 syntax
c>> 1994-10-19 DRDSFIT Krogh Changes to use M77CON
c>> 1992-11-18 DRDSFIT CLL   Changed order of arguments in DSFIT.
c>> 1992-10-29 C. L. Lawson, JPL
c     Demonstration driver for DSFIT, DSVAL, DSQUAD, DSTOP, DPVAL, DPQUAD
c     ------------------------------------------------------------------
c--D replaces "?": DR?SFIT, ?SFIT, ?SVAL, ?SQUAD, ?STOP, ?PVAL, ?PQUAD
c--&                ?PRPL
c     ------------------------------------------------------------------
c++ Code for .C. is inactive
c%%   long int k;
c++ End
      integer I, IERR, NXY, KORDER, MPC, NCOEF, NDERIV, NPC, NT, LDW
      parameter(NXY = 12, NCOEF=8, KORDER=4, NT = NCOEF+KORDER,LDW = 10)
      parameter(MPC = NCOEF-KORDER+1)
```

```
      external DSVAL, DSQUAD, DPVAL, DPQUAD
      double precision BDIF(NCOEF*KORDER), DSVAL, DSQUAD, DPVAL, DPQUAD
      double precision BCOEF(NCOEF), PCOEF(MPC*KORDER)
      double precision SD(1), SIGFAC, TKNOTS(NT), W(LDW,KORDER+1)
      double precision X(NXY), XI(MPC+1), Y(NXY), YFIT, Z
      character IMAGE*31
      data X / 2.D0, 4.D0, 6.D0, 8.D0,10.D0,12.D0,
     *         14.D0,16.D0,18.D0,20.D0,22.D0,24.D0/
      data Y /2.2D0,4.0D0,5.0D0,4.6D0,2.8D0,2.7D0,
     *         3.8D0,5.1D0,6.1D0,6.3D0,5.0D0,2.0D0/
      data TKNOTS / 4*2.0D0, 6.4D0, 10.8D0, 15.2D0, 19.6D0, 4*24.0D0 /
      data NDERIV / 0 /
      data SD(1) / -1.0D0 /
c     ------------------------------------------------------------------
      print'('' DRDSFIT''/ '' Demo driver for DSFIT, DSVAL, DSQUAD,'',
     *  '' DSTOP, DPVAL, DPQUAD'')'
      print'(/'' KORDER ='',i3,'', NCOEF ='',i3)', KORDER, NCOEF
c++ Code for ~.C. is active
      print'('' TKNOTS() = '',4f10.5/(14x,4f10.5))',
     *   (TKNOTS(I), I = 1, NT)
c++ Code for .C. is inactive
c%%     printf( "\n   TKNOTS() = ");
c%%     for (i = 1; i <= NT+3; i+=4){
c%%        for (k = i; k <= min( i+3, NT ); k++)
c%%           printf( "%10.5f", Tknots[k] );
c%%        if (i + 3 < NT) printf( "\n               ");}
c++ End
      call DSFIT(X, Y, SD, NXY, KORDER, NCOEF, TKNOTS, BCOEF,
     *           SIGFAC,IERR, LDW, W)
c++ Code for ~.C. is active
      print'(/'' After call to DSFIT:''/''  IERR ='',i5,
     *  '', SIGFAC ='', f10.5 //''  BCOEF() = '',
     *  4f10.5/(13x,4f10.5))', IERR, SIGFAC, (BCOEF(I),I=1,NCOEF)
c++ Code for .C. is inactive
c%%     printf( "\n After call to DSFIT:\n   IERR =%5ld,  SIGFAC ="
c%%        "%10.5f\n\n   BCOEF() = ", ierr, sigfac);
c%%     for (i = 1; i <= NCOEF+3; i+=4){
c%%        for (k = i; k <= min( i+3, NCOEF ); k++)
c%%           printf( "%10.5f", Bcoef[k] );
c%%        if (i + 3 < NCOEF) printf( "\n               ");}
c++ End
      print'(/'' Evaluating fitted spline function using DSVAL:'')'
      print'(/
     * ''    I     X        Y        YFIT   R=Y-YFIT                YFIT''/)'

      do 20 I=1,NXY
       YFIT=  DSVAL(KORDER, NCOEF, TKNOTS, BCOEF, X(I), NDERIV)
       call DPRPL(YFIT, '*', IMAGE, 31, 1.9d0, 6.3d0, .true.)
       print'(3x,i2,2f9.3,f10.3,3x,a31)',
     *    I, X(I), Y(I), YFIT, Y(I)-YFIT, IMAGE
   20 continue

      Z = DSQUAD(KORDER, NCOEF, TKNOTS, BCOEF, 5.0d0, 20.0d0)
      print'(/'' Integral from 5.0  to 20.0 using DSQUAD:'',f12.5)',Z

      call DSTOP(KORDER, NCOEF, TKNOTS, BCOEF, BDIF, NPC, XI, PCOEF)
      print'(/
     * '' Using DSTOP to convert from B-spline basis to power basis.'')'
      print'('' NPC ='',i3)',NPC
```

```
c++ Code for ~.C. is active
      print'('' XI() = '',4f10.5/(14x,4f10.5))',(XI(I),I=1,NPC+1)
      print'('' PCOEF() = '',4f10.5/(14x,4f10.5))',
     *   (PCOEF(I),I=1,NPC*KORDER)
c++ Code for .C. is inactive
c%%   printf( "       XI() = ");
c%%   for (i = 1; i <= (npc + 1); i+=4){
c%%      for (k = i; k <= min( i+3, npc+1 ); k++)
c%%           printf( "%10.5f", Xi[k] );
c%%      if (i <= npc) printf( "\n                ");}
c%%   printf( "\n    PCOEF() = ");
c%%   for (i = 1; i <= (npc*KORDER); i+=4){
c%%      for (k = i; k <= min( i+3, npc*KORDER ); k++)
c%%           printf( "%10.5f", Pcoef[k] );
c%%      if (i < npc*KORDER) printf( "\n                ");}
c++ End
      print'(/'' Evaluating fitted spline function using DPVAL:'')'
      print'(/
     *  ''    I     X        Y        YFIT   R=Y-YFIT                YFIT''/)'

      do 40 I=1,NXY
       YFIT= DPVAL(KORDER, NPC, XI, PCOEF, X(I), NDERIV)
       call DPRPL(YFIT, '*', IMAGE, 31, 1.9d0, 6.3d0, .true.)
       print'(3x,i2,f6.0,2f9.3,f10.3,3x,a31)',
     *    I, X(I), Y(I), YFIT, Y(I)-YFIT, IMAGE
   40 continue

      Z = DPQUAD(KORDER, NPC, XI, PCOEF, 5.0d0, 20.0d0)
      print'(/'' Integral from 5.0  to 20.0 using DPQUAD:'',f12.5)',Z
      end
```

ODDSFIT

```
DRDSFIT
Demo driver for DSFIT, DSVAL, DSQUAD, DSTOP, DPVAL, DPQUAD

  KORDER =  4,  NCOEF =  8
  TKNOTS() =    2.00000   2.00000   2.00000   2.00000
                6.40000  10.80000  15.20000  19.60000
               24.00000  24.00000  24.00000  24.00000

After call to DSFIT:
  IERR =    0,  SIGFAC =   0.14664

  BCOEF() =    2.20672   3.33355   7.10955   0.91845
               4.88398   7.24971   5.03117   1.99475

Evaluating fitted spline function using DSVAL:

   I     X        Y        YFIT   R=Y-YFIT                YFIT

   1     2.     2.200     2.207     -0.007      *
   2     4.     4.000     3.958      0.042                  *
   3     6.     5.000     5.111     -0.111                           *
   4     8.     4.600     4.430      0.170                     *
   5    10.     2.800     2.959     -0.159            *
```

```
   6    12.     2.700     2.646      0.054          *
   7    14.     3.800     3.734      0.066                  *
   8    16.     5.100     5.162     -0.062                           *
   9    18.     6.100     6.132     -0.032                                    *
  10    20.     6.300     6.233      0.067                                     *
  11    22.     5.000     5.033     -0.033                           *
  12    24.     2.000     1.995      0.005     *

Integral from 5.0  to 20.0 using DSQUAD:     66.54641

Using DSTOP to convert from B-spline basis to power basis.
       NPC =  5
      XI() =    2.00000   6.40000  10.80000  15.20000
               19.60000  24.00000
   PCOEF() =    2.20672   0.76829   0.11795  -0.03213
                5.13370  -0.05990  -0.30617   0.04307
                2.61122  -0.25290   0.26231  -0.02300
                4.61735   0.71946  -0.04132  -0.00801
                6.30079  -0.10933  -0.14704  -0.01148

Evaluating fitted spline function using DPVAL:

   I     X       Y       YFIT   R=Y-YFIT                YFIT

   1     2.     2.200     2.207     -0.007      *
   2     4.     4.000     3.958      0.042                *
   3     6.     5.000     5.111     -0.111                         *
   4     8.     4.600     4.430      0.170                      *
   5    10.     2.800     2.959     -0.159            *
   6    12.     2.700     2.646      0.054          *
   7    14.     3.800     3.734      0.066                  *
   8    16.     5.100     5.162     -0.062                           *
   9    18.     6.100     6.132     -0.032                                    *
  10    20.     6.300     6.233      0.067                                     *
  11    22.     5.000     5.033     -0.033                           *
  12    24.     2.000     1.995      0.005     *

Integral from 5.0  to 20.0 using DPQUAD:     66.54641
```

DRDSFITC

```
c     program DRDSFITC
c>> 2001-07-16 DRDSFITC Krogh  Added exponent 0 to some constants.
c>> 1996-07-11 DRDSFITC Krogh  Special code for C conversion.
c>> 1996-05-28 DRDSFITC Krogh  Changed Fortran 90 code & changes for C.
c>> 1994-10-19 DRDSFITC Krogh  Changes to use M77CON
c>> 1993-01-13 DRDSFITC C. L. Lawson, JPL
c>> 1992-11-10 C. L. Lawson, JPL
c>> 1992-11-04 C. L. Lawson, JPL
c>> 1989-03-02 C. L. Lawson, JPL
c>> 1988-04-01 C. L. Lawson, JPL
c     DRDSFITC..  Demo driver for DSFITC, Spline fit with constraints.
c     The problem has 24 data points and 10 constraints.
c     The spline is order 4 with 9 coefficients.
c     ------------------------------------------------------------------
c--D replaces "?": DR?SFITC, ?SFITC, ?SVAL, ?SVALA, ?PRPL, ?SDIF
c     ------------------------------------------------------------------
```

```
c++ Code for .C. is inactive
c%%   long int k;
c%%#define MT      (NCOEF+KORDER)
c++ End
      integer I, KORDER, KPRINT, MXY, MT
      integer NCOEF, NDATA, NINFO, NWORK
      parameter( NDATA = 24, MXY = NDATA+10)
      parameter( NINFO = 41, NWORK = 843)
      parameter( NCOEF=9, KORDER = 4, MT = NCOEF+KORDER)
      parameter( KPRINT = 0)
      integer INFO(NINFO), ISET(3)
      external DSVAL
      double precision DSVAL
      double precision BCOEF(NCOEF), BDIF(NCOEF*3), DELX, RNORM
      double precision SDI(MXY), SMAX, SMIN, SVALUE(0:2)
      double precision TKNOTS(NCOEF+KORDER)
      double precision WORK(NWORK), X, XI(MXY), YI(MXY), YFIT
      character CCODE(MXY+1)*4, IMAGE*49
      data TKNOTS / 4*0.0D0, 1.5D0, 2.5D0, 3.3D0, 4.0D0, 4.7D0, 4*6.0D0/
      data CCODE /  24*'10~a',
     *             '10=a', '11>a', '12>a', '12>a', '12>a',
     *             '12<a', '12<a', '12<a', '11>a', '10=a',
     *             '   !'/
      data XI / 0.0D0, 0.3D0, 0.7D0, 1.0D0, 1.3D0, 1.7D0, 2.0D0, 2.3D0,
     *          2.5D0, 2.6D0, 2.8D0, 2.9D0, 3.0D0, 3.1D0, 3.2D0, 3.5D0,
     *          3.7D0, 4.0D0, 4.3D0, 4.7D0, 5.0D0, 5.3D0, 5.7D0, 6.0D0,
     *          0.0D0, 0.0D0, 0.0D0, 1.5D0, 2.5D0,
     *          3.5D0, 4.5D0, 6.0D0, 6.0D0, 6.0D0/

      data YI /  1.0D0, 1.1D0,0.9D0,1.02D0, 1.2D0,1.0D0,1.2D0,1.4D0,
     *          1.76D0, 2.0D0,2.4D0, 2.6D0, 3.0D0,3.4D0,3.7D0,4.3D0,
     *          4.45D0,4.76D0,4.8D0, 5.0D0,4.96D0,4.9D0,4.9D0,5.0D0,
     *          1.0D0,0.0D0,0.0D0,0.0D0,0.0D0,
     *          0.0D0,0.0D0,0.0D0,0.0D0,5.0D0/

      data SDI(1) / -1.0D0 /
      data ISET  / NINFO, NWORK, KPRINT /
c     ------------------------------------------------------------------
      print'('' DRDSFITC..  Demo driver for DSFITC''/'' Least-squares'',
     *  '' polynomial spline fit to data with constraints.'')'
c
      print'(/
     *  ''   I   kind   deriv   relop  active       X          Y'')'
      do 10 I = 1,MXY
         print'(1x,i3,3x,a1,7x,a1,7x,a1,7x,a1,f12.3,f10.3)', I,
     *     CCODE(I)(1:1),CCODE(I)(2:2),CCODE(I)(3:3),CCODE(I)(4:4),
     *     XI(I),YI(I)
   10 continue
      I = MXY+1
      print'(1x,i3,3x,a1,7x,a1,7x,a1,7x,a1)', I,
     *     CCODE(I)(1:1),CCODE(I)(2:2),CCODE(I)(3:3),CCODE(I)(4:4)
      print'(/'' KORDER ='',i3,'',  NCOEF ='',i3)', KORDER, NCOEF
c++ Code for ~.C. is active
      print'('' TKNOTS() = '',4f10.5/(14x,4f10.5))',
     *  (TKNOTS(I), I = 1, MT)
c++ Code for .C. is inactive
c%%     printf( "\n  TKNOTS() = " );
c%%     for (i = 1; i <= MT+3; i+=4){
c%%        for (k = i; k <= min( i+3, MT ); k++)
```

```
c%%            printf( "%10.5f", Tknots[k] );
c%%         if (i + 3 < MT) printf( "\n                 ");}
c%%      printf( "\n" );
c++ End
c
      call DSFITC(CCODE, XI, YI, SDI, KORDER, NCOEF, TKNOTS,
     *                  BCOEF, RNORM, ISET, INFO, WORK)
c
      print'(/'' After call to DSFITC:'')'
      print'(/3x,''IERR5 ='',i6,'',   NEED1 ='',i7,'',   NEED2 ='',i7/
     * 3x,''M1    ='',i6,'',   MFIT  ='',i7,'',   NS    ='',i7/
     * 3x,''RNORM ='',f12.5)', INFO(1),INFO(2),INFO(3),INFO(4),INFO(5),
     * INFO(6), RNORM
c++ Code for ~.C. is active
      print'(/''  BCOEF() = '',4f10.5/(13x,4f10.5))',
     *   (BCOEF(I),I=1,NCOEF)
c++ Code for .C. is inactive
c%%      printf( "\n  BCOEF() = " );
c%%      for (i = 1; i <= NCOEF+3; i+=4){
c%%         for (k = i; k <= min( i+3, NCOEF ); k++)
c%%            printf( "%10.5f", Bcoef[k] );
c%%         if (i + 3 < NCOEF) printf( "\n                 ");}
c%%      printf( "\n" );
c++ End
c
      call DSDIF(KORDER, NCOEF, TKNOTS, BCOEF, 2, BDIF)
      SMIN = 0.0d0
      SMAX = 0.0d0
      DELX = (XI(NDATA)-XI(1))/30.0D0
      X = XI(1)
      do 20 I = 0, 31
         call DSVALA(KORDER, NCOEF, TKNOTS, 2, BDIF, X, SVALUE)
         SMIN = min(min(SMIN, SVALUE(0)), min(SVALUE(1), SVALUE(2)))
         SMAX = max(max(SMAX, SVALUE(0)), max(SVALUE(1), SVALUE(2)))
         X = X + DELX
   20 continue
      print'(/a)',
     * '     X    YFIT  YFIT''  YFIT''''   '
      X = XI(1)
      do 40 I=0,31
         call DSVALA(KORDER, NCOEF, TKNOTS, 2, BDIF, X, SVALUE)
         IMAGE = ' '
         call DPRPL(SVALUE(0), '*', IMAGE, 49, SMIN, SMAX, .false.)
         call DPRPL(SVALUE(1), '1', IMAGE, 49, SMIN, SMAX, .false.)
         call DPRPL(SVALUE(2), '2', IMAGE, 49, SMIN, SMAX, .false.)
         print'(1x,f6.3,f7.3,f7.3,f7.3,1x,a49)',X, SVALUE(0), SVALUE(1),
     *      SVALUE(2), IMAGE
         X = X + DELX
   40 continue
c                          Compute and print residuals.
      print'(/'' Residuals at the data points:''//
     *   ''    I   XI(I)   YI(I)    YFIT   YFIT-YI(I)'',
     *   ''           YFIT-YI(I)'')'
      do 60 I = 1, NDATA
         YFIT = DSVAL(KORDER, NCOEF, TKNOTS, BCOEF, XI(I), 0)
         call DPRPL(YFIT-YI(I), '*', IMAGE, 39, -0.18d0, 0.18d0, .true.)
         print'(1x,i4,f8.3,f8.3, f8.3, f10.3,1x,a39)',
     *      I, XI(I), YI(I), YFIT, YFIT-YI(I), IMAGE(1:39)
   60 continue
```

end

ODDSFITC

```
DRDSFITC..  Demo driver for DSFITC
Least-squares polynomial spline fit to data with constraints.

  I   kind   deriv   relop  active       X         Y
  1     1       0       ~       a       0.000     1.000
  2     1       0       ~       a       0.300     1.100
  3     1       0       ~       a       0.700     0.900
  4     1       0       ~       a       1.000     1.020
  5     1       0       ~       a       1.300     1.200
  6     1       0       ~       a       1.700     1.000
  7     1       0       ~       a       2.000     1.200
  8     1       0       ~       a       2.300     1.400
  9     1       0       ~       a       2.500     1.760
 10     1       0       ~       a       2.600     2.000
 11     1       0       ~       a       2.800     2.400
 12     1       0       ~       a       2.900     2.600
 13     1       0       ~       a       3.000     3.000
 14     1       0       ~       a       3.100     3.400
 15     1       0       ~       a       3.200     3.700
 16     1       0       ~       a       3.500     4.300
 17     1       0       ~       a       3.700     4.450
 18     1       0       ~       a       4.000     4.760
 19     1       0       ~       a       4.300     4.800
 20     1       0       ~       a       4.700     5.000
 21     1       0       ~       a       5.000     4.960
 22     1       0       ~       a       5.300     4.900
 23     1       0       ~       a       5.700     4.900
 24     1       0       ~       a       6.000     5.000
 25     1       0       =       a       0.000     1.000
 26     1       1       >       a       0.000     0.000
 27     1       2       >       a       0.000     0.000
 28     1       2       >       a       1.500     0.000
 29     1       2       >       a       2.500     0.000
 30     1       2       <       a       3.500     0.000
 31     1       2       <       a       4.500     0.000
 32     1       2       <       a       6.000     0.000
 33     1       1       >       a       6.000     0.000
 34     1       0       =       a       6.000     5.000
 35                              !

 KORDER =  4,  NCOEF =  9
 TKNOTS() =     0.00000   0.00000   0.00000   0.00000
                1.50000   2.50000   3.30000   4.00000
                4.70000   6.00000   6.00000   6.00000
                6.00000

After call to DSFITC:

 IERR5 =     0,   NEED1 =     41,   NEED2 =    843
 M1    =    10,   MFIT  =     24,   NS    =      8
 RNORM =     0.37206

 BCOEF() =     1.00000   1.01613   1.04300   1.07848
               4.07150   4.87706   4.92040   4.96864
               5.00000

    X     YFIT   YFIT'   YFIT''
0.000   1.000   0.032   0.000                    2       *
```

0.200	1.006	0.032	0.000	
0.400	1.013	0.032	0.000	
0.600	1.019	0.032	0.000	
0.800	1.026	0.032	−0.000	
1.000	1.032	0.032	−0.000	
1.200	1.039	0.032	−0.000	
1.400	1.045	0.032	−0.000	
1.600	1.052	0.052	0.395	
1.800	1.076	0.210	1.186	
2.000	1.147	0.527	1.977	
2.200	1.297	1.001	2.768	
2.400	1.558	1.634	3.559	
2.600	1.959	2.360	3.045	
2.800	2.480	2.787	1.225	
3.000	3.049	2.850	−0.595	
3.200	3.595	2.549	−2.414	
3.400	4.047	1.943	−3.064	
3.600	4.377	1.382	−2.543	
3.800	4.606	0.926	−2.022	
4.000	4.755	0.573	−1.500	
4.200	4.842	0.317	−1.065	
4.400	4.887	0.147	−0.629	
4.600	4.907	0.065	−0.194	
4.800	4.919	0.059	0.022	
5.000	4.931	0.063	0.019	
5.200	4.944	0.066	0.015	
5.400	4.957	0.069	0.011	
5.600	4.971	0.071	0.007	
5.800	4.986	0.072	0.004	
6.000	5.000	0.072	0.000	
6.200	5.014	0.072	−0.004	

Residuals at the data points:

I	XI(I)	YI(I)	YFIT	YFIT–YI(I)	YFIT–YI(I)
1	0.000	1.000	1.000	0.000	
2	0.300	1.100	1.010	−0.090	
3	0.700	0.900	1.023	0.123	
4	1.000	1.020	1.032	0.012	
5	1.300	1.200	1.042	−0.158	
6	1.700	1.000	1.060	0.060	
7	2.000	1.200	1.147	−0.053	
8	2.300	1.400	1.412	0.012	
9	2.500	1.760	1.740	−0.020	
10	2.600	2.000	1.959	−0.041	
11	2.800	2.400	2.480	0.080	
12	2.900	2.600	2.763	0.163	
13	3.000	3.000	3.049	0.049	
14	3.100	3.400	3.330	−0.070	
15	3.200	3.700	3.595	−0.105	
16	3.500	4.300	4.226	−0.074	
17	3.700	4.450	4.503	0.053	
18	4.000	4.760	4.755	−0.005	
19	4.300	4.800	4.869	0.069	
20	4.700	5.000	4.913	−0.087	
21	5.000	4.960	4.931	−0.029	
22	5.300	4.900	4.950	0.050	
23	5.700	4.900	4.978	0.078	
24	6.000	5.000	5.000	−0.000	