Abstract:

A compact and efficient algorithm for the calculation of least squares splines is presented. Intended for use in interactive design of open and closed free-form curves, the algorithm performs parts of the computation in parallel to enable real-time curve-fitting. It employs the B-spline basis to form the normal equations and solves the normal equations by an envelope of $L D L^T$ factorization. A FORTRAN IV implementation is included.

A Real-Time Algorithm
for Least Squares Splines
and Its Application in
Computer-Aided Geometric Design

S. C. Eisenstat, J. W. Lewis, M. H. Schultz

Research Report #29

March 1975

This work was partially supported by ONR grant N0014-67-A-0097-0016 and NSF grant GJ-43157. The results were presented at the Conference for Computer Aided Geometric Design, University of Utah, May 1974.

1. *Introduction*

The creation and manipulation of free-form curves is one of the fundamental problems of computer aided geometric design. For objects such as ship hulls, automobile bodies, shoe lasts, or television tubes there are no canonical shapes: the form of the object is unrestricted. A design system must enable the designer to create, rapidly and accurately, the curves which describe these objects.

In many design systems, measurements from the designer's sketch specify the desired curve. From those measurements the design system should compute a compact and faithful approximation to that curve. Too often this approximation will include spurious oscillations not found in the data, yet may not reproduce cusps and inflection points that do exist in the data.

To overcome these difficulties, a combination of spline regression and B-spline Bezier curves may be employed [12]. The least squares spline provides a good initial approximation to the drawn curve and the Bezier polygon for this spline curve provides a convenient means for correcting defects in the approximation and for changing the curve. The theory and applications of Bezier curves are described elsewhere [8,10,12]; here the goal is an algorithm for the calculation of least squares splines in interactive design systems.

In an interactive system the curve data are obtained from a graphics input device (typically a tablet) while the designer draws the curve. In such a system the least squares approximation to the curve should be displayed within seconds of the curve's completion, and if possible, that fit should be displayed while the curve is being drawn. In addition, the fitting program should be small enough for a local graphics processor.

This paper describes a least squares algorithm which meets these goals. Parts of the computation are overlapped to achieve fast response and to enable real-time curve-fitting. To minimize memory requirements only non-zero elements of the least squares equations are stored and the data points themselves are not stored at all. A well-documented FORTRAN IV implementation of the algorithm is included as Appendix I.

The remainder of the paper is divided into five sections. Sections 2 and 3 review, respectively, the properties of parametric spline curves and the development of the normal equations. In Section 4 the details of an efficient algorithm for the least squares spline calculation are described and the computational requirements for that algorithm are estimated. In Section 5, using the local convergence property of least squares splines, the algorithm is re-structured as a pipeline, enabling real-time computation of the least squares spline and reducing storage requirements over the algorithm of Section 4. In the final section the algorithm is modified further for periodic curve fitting and the additional computational cost over the non-periodic curve fit is estimated.

2. *Spline Curves*

Polynomial splines are one of many possible generalizations of piecewise linear functions. Just as the piecewise linear spline is the function of least length connecting a set of points, so the cubic spline is the function of minimum strain energy connecting a set of points and having fixed slopes as its end points (Figure 2.1). From analogous minimum principles, higher order, odd degree polynomial splines and other, more general splines may be defined [17,18,20]. The cubic spline behaves much like the draftsman's spline -- a thin piece of bamboo held down by lead ducks -- for which it was named. Both cubic and draftsman's splines are widely used for approximating curves.

Alternatively, splines may be considered as piecewise polynomials -- a set of polynomials in the intervals of a partition joined so that derivatives match. For $k \geq 2$, the $(k-1)$ degree polynomial spline $S_k^\Delta(t)$ defined over the uniform knot set

(2.1) Δ: $0, h, \ldots, m \cdot h, (m+1) \cdot h = a$

is a polynomial of degree $(k-1)$ in each interval $(i \cdot h, (i+1) \cdot h)$ and has $(k-2)$ continuous derivatives over the complete interval $[0,a]$. This definition may be generalized further to allow non-uniform knots, but for efficiency and simplicity, the algorithms presented in this paper are

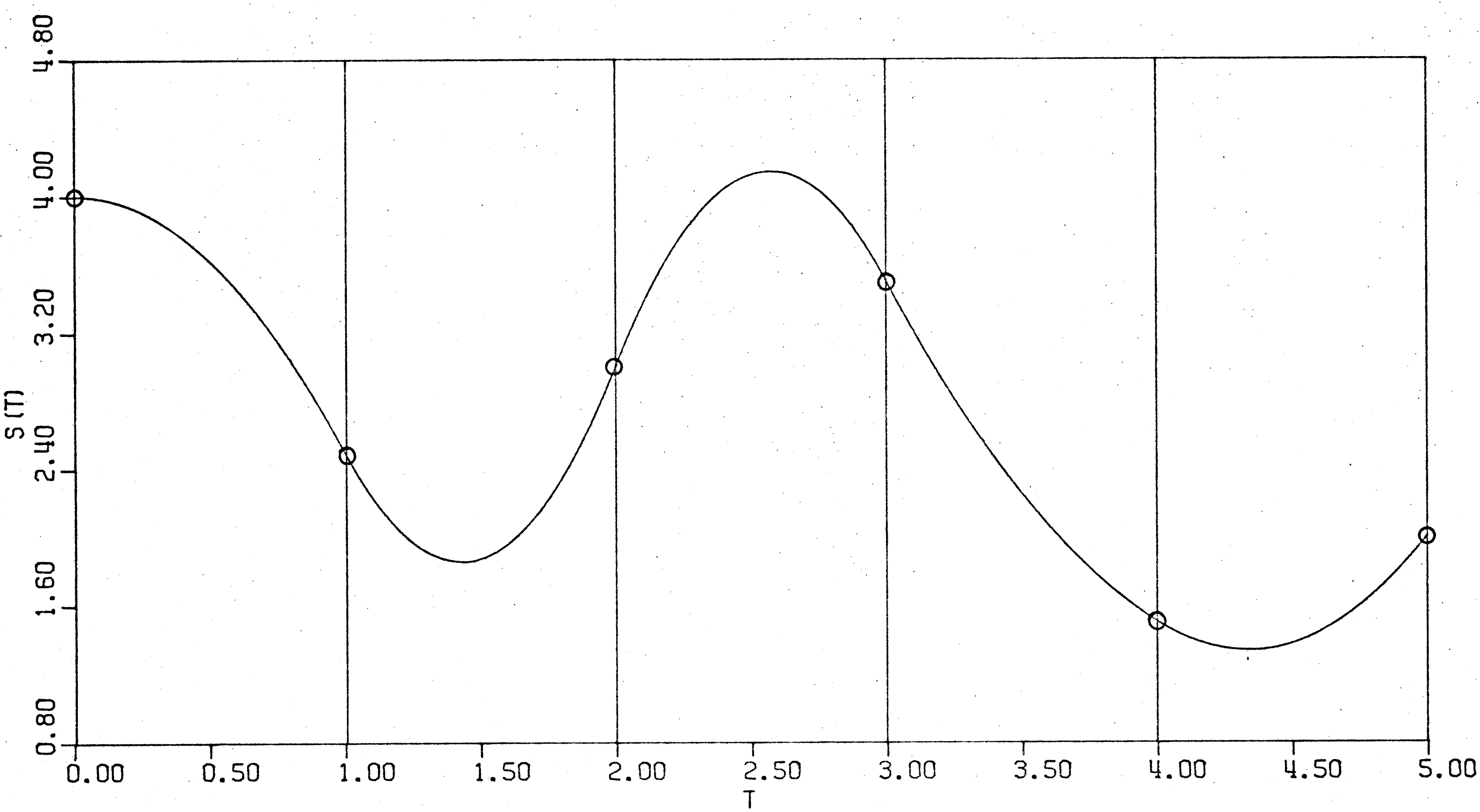


Figure 2.1: Cubic spline. The vertical lines indicate knot locations and the circles are the points which, with two end conditions, define the spline.

developed for uniform knots. The development for non-uniform knots is equally straightforward [6].

Since the spline is a set of simple polynomials, the spline may be evaluated from its polynomial coefficients

(2.2) $\{C_{i\ell}: 0 \le i \le m, 0 \le \ell \le k-1\}$,

where for t satisfying

(2.3) $i \cdot h \le t < (i+1) \cdot h$,

the value of the spline is given by

(2.4) $$S(t) = \sum_{\ell=0}^{k-1} C_{i\ell}(t-ih)^{\ell}.$$

A third convenient characterization of a polynomial spline is in terms of the B-spline basis [1,2,3,4]. Any polynomial spline may be written as a linear combination

(2.5) $$S(t) = \sum_{j=1}^{m+k} A_j N_{jk}(t)$$

of the B-spline basis functions N_{jk} (see Figure 2.2). The A_j are the m+k B-spline basis coefficients.

The B-spline basis functions are themselves splines, and like any other spline, may be written as piecewise polynomials (see Appendix II). The B-splines are non-negative,

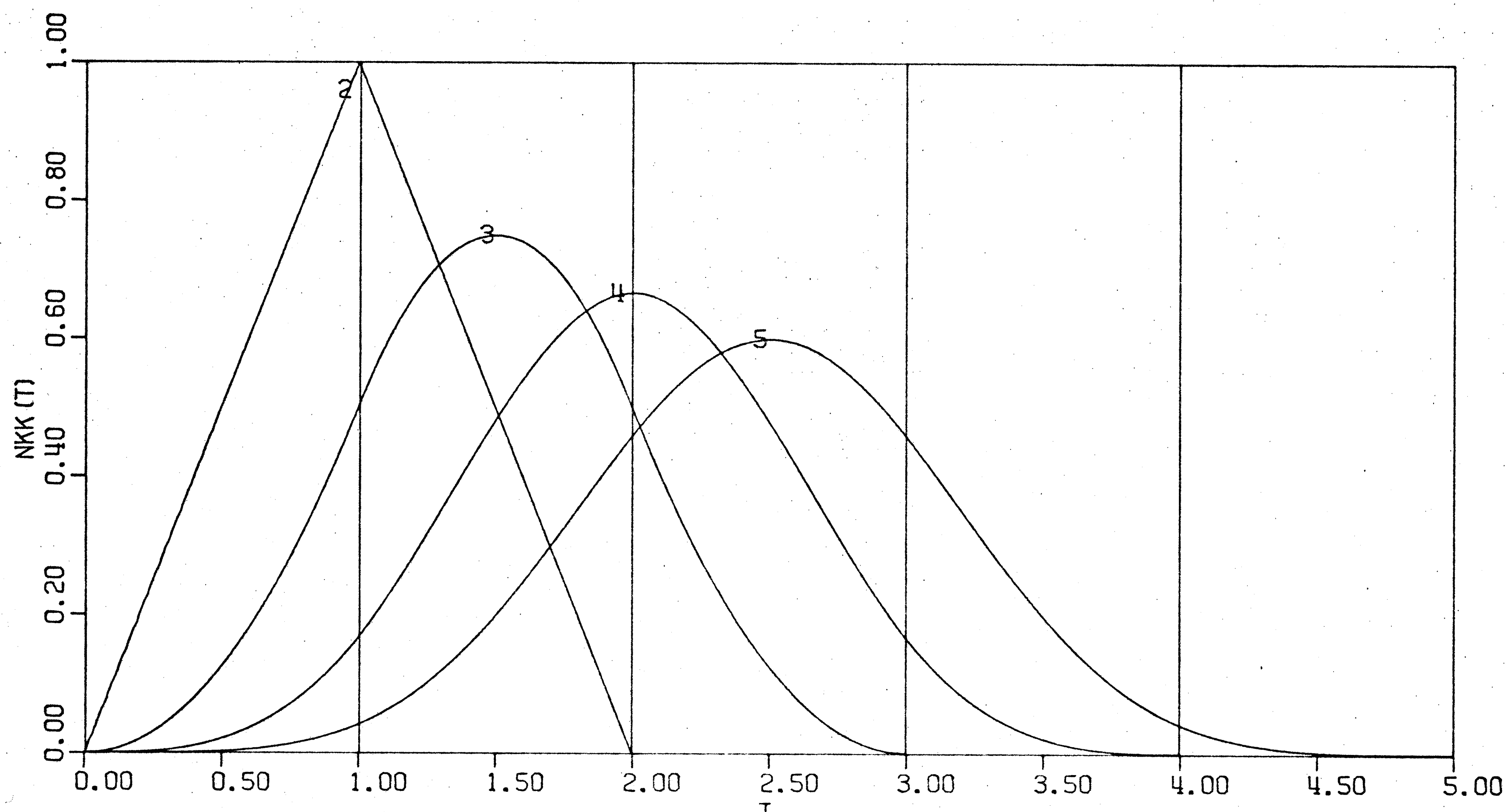


Figure 2.2: B-splines for k = 2,3,4,5. The vertical lines indicate knot locations.

(2.6) $N_{jk}(t) \geq 0$ for $0 \leq t \leq a$,

normalized,

(2.7) $$1 = \sum_{j=1}^{m+k} N_{jk}(t) \text{ for } 0 \leq t \leq a,$$

and have local support,

for $0 \leq t \leq a$ and $1 \leq j \leq m+k$

(2.8) $N_{jk}(t) \neq 0$ if and only if $(j-k)\cdot h < t < j\cdot h$.

Since the B-spline basis is local, for

(2.9) $i\cdot h \leq t < (i+1)\cdot h$

the sum (2.5) reduces to

(2.10) $$S(t) = \sum_{j=i+1}^{i+k} A_j N_{jk}(t).$$

As an example of the three spline definitions, consider the following three different characterizations of a simple piecewise linear spline (Figure 2.1).

1) The function of least arc length connecting the points (0,2), (1,5), and (2,1).

2) The piecewise polynomial

(2.11) $$S(t) = \left\{ \begin{matrix} 2 + t\cdot 3 & \text{for } 0 \leq t < 1 \\ 5 + (1-t)\cdot 4 & \text{for } 1 \leq t \leq 2 \end{matrix} \right\}.$$

3) The linear combination of B-spline basis functions

(2.12) $S(t) = 2 \cdot N_{12}(t) + 5 \cdot N_{22}(t) + 1 \cdot N_{32}(t)$

where

(2.13) $N_{12}(t) = \begin{cases} 1-t & \text{for } 0 \le t \le 1 \\ 0 & \text{otherwise} \end{cases}$

(2.14) $N_{22}(t) = \begin{cases} t & \text{for } 0 \le t < 1 \\ 2-t & \text{for } 1 \le t < 2 \\ 0 & \text{otherwise} \end{cases}$

(2.15) $N_{32}(t) = \begin{cases} t-1 & \text{for } 1 \le t \le 2 \\ 0 & \text{otherwise} \end{cases}$.

So far the discussion has been restricted to spline functions of one variable. "Spline curves" in two or more dimensions may be represented as two or more spline functions of a parameter t. In two dimensions, a parametric spline curve is given by

(2.16) $\underset{\sim}{S}(t) = (S_x(t), S_y(t)) \quad \text{for } 0 \le t \le a$,

where $S_x(t)$ and $S_y(t)$ are one-dimensional splines.

For t satisfying

(2.17) $i \cdot h \le t < (i+1) \cdot h$

the parametric spline may be written as a polynomial with vector coefficients

(2.18) $\underset{\sim}{S}(t) = \sum_{\ell=0}^{k-1} \underset{\sim}{C}_{i\ell} \, (t - i \cdot h)^{\ell}$

or as a linear combination of B-spline basis functions

$$(2.19)\quad \underset{\sim}{S}(t) = \sum_{j=i+1}^{i+k} \underset{\sim}{A}_j N_{jk}(t)$$

where the $\underset{\sim}{A}_i$ are the vector B-spline basis coefficients.

3. *Least Squares*

Given measurements from the designer's drawn curve, the proposed design system must produce a spline curve which is in some sense "close" to the original. Furthermore, the resulting spline curve should have relatively few parameters so that it may be stored compactly and so that the Bezier polygon for the curve will have a small number of vertices [15].

One means of computing such a curve is least squares. Unlike an interpolate, the least squares spline need not pass through every data point; therefore it may have far fewer parameters than there are data. It will tend to smooth measurement noise, to flatten spurious "wiggles," and to be insensitive to gaps in the data [11]. In addition, as the following sections show, the least squares spline may be computed quite efficiently.

In one dimension the computation of the least squares spline is particularly simple. Given a set of weighted data

$$(3.1)\quad X = \{(t_q,\, y_q,\, w_q) : 1 \le q \le M\},$$

with abscissa t_q, ordinate y_q, and positive weight w_q, the least squares spline $S_k^\Delta(t)$ is that spline closest to the data in that it minimizes the weighted least squares distance

$$(3.2)\quad d(S,X) = \sum_{q=1}^{M} w_q \cdot (S(t_q) - y_q)^2$$

over all splines of order k and knot set Δ. For simplicity in the following development, the weights w_q are taken to be unity.

The traditional approach to this approximation problem is to form and solve the normal equations. The spline is written as a linear combination of some set of basis functions (here chosen as the B-splines)

$$(3.3)\quad S(t) = \sum_{j=1}^{m+k} A_j N_{jk}(t),$$

so that the distance functional d(S,X) may be written as a quadratic function of the basis coefficients $\underset{\sim}{A}$

$$(3.4)\quad F(\underset{\sim}{A}) = d(S,X) = \underset{\sim}{A}^T \cdot G \cdot \underset{\sim}{A} - 2 \cdot \underset{\sim}{A}^T \cdot \underset{\sim}{B} + C,$$

where the discrete Gram matrix G is given by

$$(3.5)\quad G = [g_{ij}] = [\sum_{q=1}^{M} N_{ik}(t_q) \cdot N_{jk}(t_q)],$$

the vector $\underset{\sim}{B}$ is given by

$$(3.6)\quad \underset{\sim}{B} = [b_i] = [\sum_{q=1}^{M} y_q \cdot N_{ik}(t_q)],$$

and the constant C is given by

(3.7) $$C = \sum_{q=1}^{M} y_q^2 .$$

The minimization of F with respect to $\underset{\sim}{A}$ is a simple multivariate calculus problem. The unique minimum of F is attained at $\underset{\sim}{A}^*$ if the gradient

(3.8) $$\frac{\partial F}{\partial \underset{\sim}{A}} = 2\cdot(G\cdot\underset{\sim}{A} - \underset{\sim}{B})$$

vanishes and the Hessian matrix

(3.9) $$\left[\frac{\partial^2 F}{\partial A_i \partial A_j}\right] = 2\cdot G$$

is positive definite. Therefore the matrix form of the normal equations

(3.10) $$G\cdot\underset{\sim}{A}^* = \underset{\sim}{B}$$

must be satisfied at the minimum $\underset{\sim}{A}^*$.

For typical graphics tablet data, and for any other data numerous and reasonably distributed with respect to the knots, the Gram matrix G will be positive definite and a unique minimum will exist [2,11]. Furthermore, under appropriate conditions, the matrix G will be well conditioned and the numerical stability of the method is assured [1,5, 11,16].

For least squares curves in two or more dimensions, a different distance functional is minimized. This functional is constructed so that an n-dimensional problem reduces to n one-dimensional problems. The following development in two dimensions generalizes easily to curves in higher dimensions.

Given a set of data

$$(3.11)\quad Y = \{(x_q, y_q): 1 \le q \le M\},$$

the least squares spline

$$(3.12)\quad \underset{\sim}{S}_k^{\Delta}(t) = \sum_{j=1}^{m+k} (A_j^x, A_j^y) \cdot N_{jk}(t)$$

is that spline which minimizes the distance functional

$$(3.13)\quad d_2(\underset{\sim}{S},Y) = \sum_{q=1}^{M} [(S_x(t_q) - x_q)^2 + (S_y(t_q) - y_q)^2]$$

over all parametric splines of given order k and knots Δ. The parameterization of the data t_q is chosen as a piecewise linear approximation to the arc length

$$(3.14)\quad \begin{aligned} t_1 &= 0 \\ t_q &= t_{q-1} + [(x_q - x_{q-1})^2 + (y_q - y_{q-1})^2]^{\frac{1}{2}} \\ &\quad \text{for } 2 \le q \le M. \end{aligned}$$

As before, the distance functional $d_2(\underset{\sim}{S},Y)$ may be written as a quadratic function of $\underset{\sim}{A}^x$ and $\underset{\sim}{A}^y$

$$(3.15)\quad F(\underset{\sim}{A}^x, \underset{\sim}{A}^y) = \begin{array}{l} (\underset{\sim}{A}^x)^T \cdot G \cdot \underset{\sim}{A}^x - 2 \cdot (\underset{\sim}{A}^x)^T \cdot \underset{\sim}{B}^x + C^x + \\ (\underset{\sim}{A}^y)^T \cdot G \cdot \underset{\sim}{A}^y - 2 \cdot (\underset{\sim}{A}^y)^T \cdot \underset{\sim}{B}^y + C^y, \end{array}$$

where the discrete Gram matrix G is given by

$$(3.16)\quad G = [g_{ij}] = [\sum_{q=1}^{M} N_{ik}(t_q) \cdot N_{jk}(t_q)],$$

the vectors $\underset{\sim}{B}^x$ and $\underset{\sim}{B}^y$ are given by

$$(3.17)\quad \underset{\sim}{B}^x = [b_i^x] = [\sum_{q=1}^{M} x_q \cdot N_{ik}(t_q)]$$

$$(3.18)\quad \underset{\sim}{B}^y = [b_i^y] = [\sum_{q=1}^{M} y_q \cdot N_{ik}(t_q)],$$

and the constants C^x and C^y are given by

$$(3.19)\quad C^x = \sum_{q=1}^{M} x_q{}^2$$

$$(3.20)\quad C^y = \sum_{q=1}^{M} y_q{}^2.$$

At the least squares solution (A_x^*, A_y^*) the gradients

$$(3.21)\quad \frac{\partial F}{\partial \underset{\sim}{A}^x} = 2 \cdot (G \cdot \underset{\sim}{A}^x - \underset{\sim}{B}^x)$$

$$(3.22)\quad \frac{\partial F}{\partial \underset{\sim}{A}^y} = 2 \cdot (G \cdot \underset{\sim}{A}^y - \underset{\sim}{B}^y)$$

must vanish so that the normal equations

(3.23) $G \cdot \underset{\sim}{A}^x = \underset{\sim}{B}^x$

(3.24) $G \cdot \underset{\sim}{A}^y = \underset{\sim}{B}^y$

must be satisfied.

The two equations (3.23, 3.24) are identical to the one-dimensional normal equation (3.10) and may be solved separately by the techniques used for the one-dimensional problem. Note that the same matrix G appears in both equations.

4. Computational Considerations

The spline curve-fitting method of Section 3 may be implemented quite efficiently. In this section a detailed implementation of the calculation is presented and the work required is estimated. These work estimates will be used in Section 5 to determine the relative time required for various parts of the calculation.

The first problem is the evaluation of the spline itself. A polynomial spline is most efficiently evaluated from its piecewise polynomial representation (2.2-4). Given a spline of order k, with knots

(4.1) $\Delta\colon 0, h, \ldots, m\cdot h, (m+1)\cdot h = a$

and piecewise polynomial coefficients

(4.2) $\{C_{i\ell}\colon 0 \le i \le m,\ 0 \le \ell \le k-1\}$,

the spline can be computed by Horner's rule

$$S_0 = C_{i,k-1}$$

(4.3) $$S_\ell = (t - i\cdot h)\cdot S_{\ell-1} + C_{i,k-1-\ell} \quad \text{for } 1 \le \ell \le k-1$$

$$S(t) = S_{k-1}$$

for

(4.4) $i \cdot h \leq t < (i+1) \cdot h.$

For the cubic spline (k = 4) with $\delta = t - i \cdot h$, the preceding is simply

(4.5) $S_4(t) = C_{i0} + \delta \cdot (C_{i1} + \delta \cdot (C_{i2} + \delta \cdot C_{i3}))$,

requiring three multiplications. In general, the evaluation of a spline of order k requires (k-1) multiplications.

Using this method to evaluate the basis functions, the normal equations are computed. The Gram matrix G of the normal equations is symmetric by definition; and since the B-spline basis is local (see equation 2.8), the matrix is banded, i.e.

(4.6) $g_{ij} \neq 0$ if and only if $|i-j| < k$,
for $1 \leq i, j \leq m+k$.

Consequently, only the lower triangle of the band of G need be computed or stored, requiring

(4.7) $(m+k) \cdot k$ locations.

The elements of G and $\underset{\sim}{B}$ are computed in the following manner:

0) Initialize G and $\underset{\sim}{B}$ to zero. Let q = 1.

1) Determine an interval of Δ such that

(4.8) $i \cdot h \leq t_q < (i+1) \cdot h.$

2) Evaluate those basis functions

$$N_{i+1,k}(t_q),\ \ldots,\ N_{i+k,k}(t_q) \tag{4.9}$$

which can be nonzero at t_q using a table of their piecewise polynomials (Appendix II).

3) Compute the terms of sums (3.5-6) which depend on the point t_q and are not trivially zero (Figure 4.1). Add them to the appropriate elements of $\underset{\sim}{B}$ and of the lower triangle of G:

$$\begin{aligned}
g_{i+1,i+1} &= g_{i+1,i+1} + N_{i+1,k}(t_q) \cdot N_{i+1,k}(t_q) \\
g_{i+2,i+1} &= g_{i+2,i+1} + N_{i+2,k}(t_q) \cdot N_{i+1,k}(t_q) \\
&\vdots \\
g_{i+k,i+k} &= g_{i+k,i+k} + N_{i+k,k}(t_q) \cdot N_{i+k,k}(t_q) \\
b_{i+1} &= b_{i+1} + y_q \cdot N_{i+1,k}(t_q) \\
&\vdots \\
b_{i+k} &= b_{i+k} + y_q \cdot N_{i+k,k}(t_q).
\end{aligned} \tag{4.10}$$

4) Let q = q+1; if data are exhausted then quit; otherwise go to (1).

	X	X	X	X	.	.	.	.	.	.	.		X
G:	X	X	X	X	X	.	.	.	.	.	.	B:	X
	X	X	X	X	X	X	.	.	.	.	.		X
	X	X	X	X	X	X	X	.	.	.	.		X
i+1	.	X	X	X	+	+	+	+	.	.	.		+
.	.	.	X	X	+	+	+	+	X	.	.		+
.	.	.	.	X	+	+	+	+	X	X	.		+
i+k	.	.	.	.	+	+	+	+	X	X	X		+
	.	.	.	.	.	X	X	X	X	X	X		X
	.	.	.	.	.	.	X	X	X	X	X		X

Figure 4.1: For order k = 4, the elements of G and B depending on a given data point satisfying (4.8) are indicated by "+"; trivially zero elements are indicated by "."; and other non-zero elements are indicated by "X".

For an n-dimensional problem, neglecting low order terms, the formation of the normal equations requires

(4.11) $\sim M \cdot k^2$ multiplications

to evaluate the non-vanishing basis functions,

(4.12) $\sim \frac{1}{2} \cdot M \cdot k^2$ multiplications

to compute G, and

(4.13) $\sim n \cdot M \cdot k$ multiplications

to compute B. Note that these work estimates are independent of the number of knots and that the work required to compute G is independent of the dimension n.

The normal equations

(4.14) $G \cdot \underset{\sim}{A}^* = \underset{\sim}{B}$

are solved by a band $L \cdot D \cdot L^T$ factorization. Since G is positive definite and symmetric, there exists a unique factorization of G in the form

(4.15) $G = L \cdot D \cdot L^T$

where D is a positive diagonal matrix and L is a lower triangular matrix with unit diagonal [9]. If the relations [13]

$$(4.16)\quad \ell_{ij} = \Big(g_{ij} - \sum_{q=\max(1,i-k+1)}^{j-1} d_{qq} \cdot \ell_{iq} \cdot \ell_{jq}\Big)/d_{jj}$$

for $1 \le j \le i \le m+k$

$$(4.17)\quad d_{ii} = g_{ii} - \sum_{q=\max(1,i-k+1)}^{i-1} d_{qq} \cdot \ell_{iq}^{2}$$

for $1 \le i \le m+k$

are applied in a proper order, the elements of L and D may be computed from those of G. Using this factorization, the solution to the normal equations is obtained in two steps -- the forward solution for the temporary W

$$(4.18)\quad \underset{\sim}{W} = L^{-1} \cdot \underset{\sim}{B},$$

and the backward solution for least squares coefficients

$$(4.19)\quad \underset{\sim}{A}^{*} = (L^{T})^{-1} \cdot D^{-1} \cdot \underset{\sim}{W}.$$

Both steps involve the solution of simple triangular systems.

Band L D L^T factorization has three convenient properties. The diagonal of L is 1; the factor L has the same nonzero structure as the lower triangle of the matrix G; and the element g_{ij} is referenced only once and then it is used to compute ℓ_{ij} (or d_{ii} if $i = j$). Consequently the matrix G may be replaced in storage by the factors L and D <u>as they are computed</u>. To conserve storage only the lower triangles of the bands of L and G are stored. In the code of Appendix I the two matrices are stored as a vector in row order. For cubic splines (k = 4) the matrix G would be stored as

(4.20) $g_{11}\ g_{21}g_{22}\ g_{31}g_{32}g_{33}\ g_{41}g_{42}g_{43}g_{44}\ g_{52}g_{53}g_{54}g_{55}$

and later replaced in storage by the factorization

(4.21) $d_{11}\ \ell_{21}d_{22}\ \ell_{31}\ell_{32}d_{33}\ \ell_{41}\ell_{42}\ell_{43}d_{44}\ \ell_{52}\ell_{53}\ell_{54}d_{55}$.

This method requires no more storage than that needed for the lower triangle of the band of G and requires far fewer operations than dense Gaussian elimination. The resulting FORTRAN IV subroutine is not significantly larger than a Gaussian elimination routine [9]. For an n-dimensional curve fit, neglecting low-order terms, the factorization of G requires

(4.22) $\sim\frac{1}{2}\cdot(m+k)\cdot k^2$ multiplications,

the forward solve requires

(4.23) $\sim n\cdot(m+k)\cdot k$ multiplications,

and the backward solve requires

(4.24) $\sim n\cdot(m+k)\cdot k$ multiplications.

Note that the work required to solve the normal equations is independent of the amount of data and that the work required to factor G is independent of the dimension n.

Since the number of data points M is normally much greater than the number of parameters m+k, most of the work is in setting up the normal equations (4.11-13) rather than in their solution (4.22-24).

Once the basis coefficients have been computed, the spline may be evaluated for display on the terminal. For computational efficiency the spline should be evaluated from its piecewise polynomial representation, but it was computed in the B-spline representation. Given t and i satisfying

$$(4.25)\quad i\cdot h \le t < (i+1)\cdot h,$$

the B-spline representation of the spline is given by (equation 2.10)

$$(4.26)\quad S(t) = \sum_{j=i+1}^{i+k} A_j N_{jk}(t)$$

and the piecewise polynomial for the B-spline is given by (Appendix II)

$$(4.27)\quad N_{jk}(t) = \sum_{\ell=0}^{k-1} C^N_{j-i,\ell} \cdot (t - i\cdot h)^\ell \quad \text{for } i+1 \le j \le i+k$$

Consequently the coefficients of the piecewise polynomial can be computed from

$$(4.28)\quad C_{i\ell} = \sum_{j=i+1}^{i+k} C^N_{j-i,\ell} \cdot A_j \quad \text{for } 0 \le i \le m,\ 0 \le \ell \le k-1.$$

Neglecting lower order terms, the conversion of the entire spline requires

(4.29) $\sim m \cdot k^2$ multiplications.

5. *Real Time Calculations*

The algorithm of the previous section is neither real-time nor compact. For the sample "Splines" curve and fit of Figure 5.1 (order k = 4, dimension n = 2, number of data points M = 2500, and number of knots m = 46), the calculation requires ten seconds of PDP-10 processor time, 3·2500 locations for data storage, 6·50 locations for storage of the least squares arrays G and $\underset{\sim}{B}$, and 4·47 locations for storage of the piecewise polynomial. Clearly, both speed and storage requirements are excessive for a local graphics processor.

With small modifications, the algorithm can be made both compact and real-time. Storage requirements are reduced by eliminating storage of the data and of unneeded elements in the least squares matrices, and real-time response is achieved by performing nearly all of the least squares calculation while the curve is being drawn. In fact the fit for the first part of the curve can be displayed <u>before the entire curve has been drawn</u>, requiring somewhat more computation but far less storage than for a sequential implementation.

The calculation divides naturally into three major processes -- the formation of the normal equations, the factorization and forward solution of the normal equations,

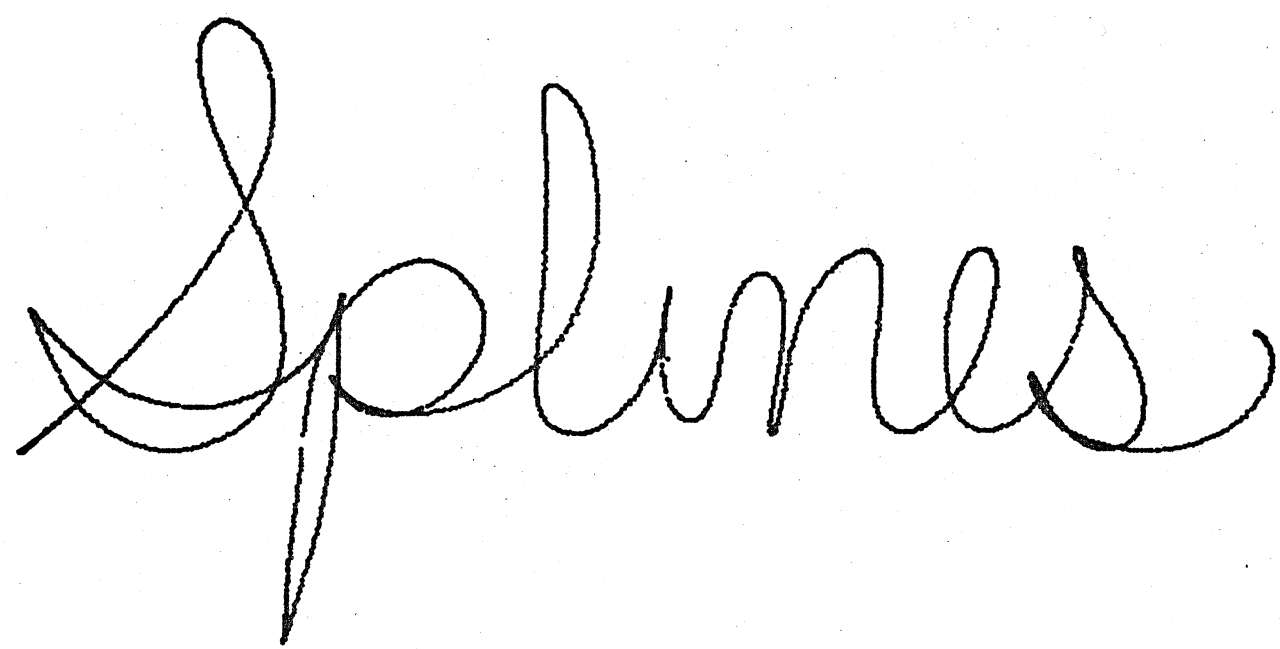


Figure 5.1a: "Splines" data 2500 discrete points taken from a ten bit resolution graphics tablet.

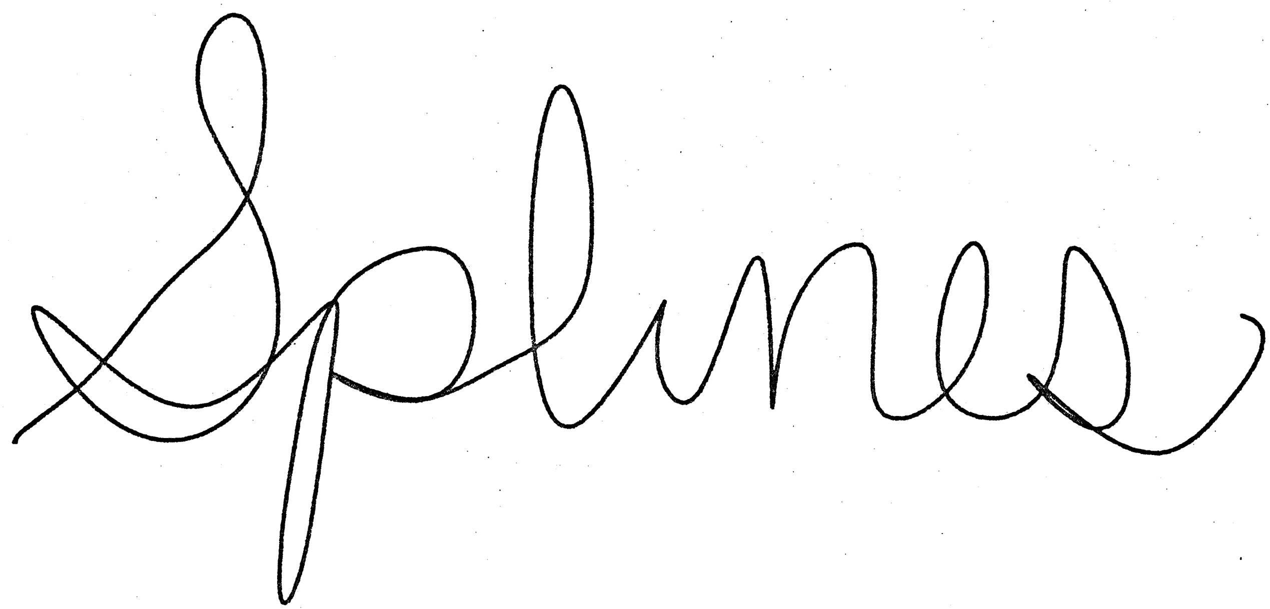


Figure 5.1b: Cubic spline fit to "Splines" data, m = 46.

and the back solution for the basis coefficients and their conversion to a piecewise polynomial. These processes need not be performed sequentially. In this section they are pipe-lined so that, at any given point in time, each process will have been completed to the greatest extent possible. In this way the idle time during curve drawing is employed to display the fit as soon as possible.

Of the three processes, the formation of the normal equations requires the most processor time (Table 5.1).

1) Normal equations	$\sim M \cdot k \cdot (3k/2 + n)$ multiplications ~ 80000
2) Factorization and forward solve	$\sim (m+k) \cdot k \cdot (k/2 + n)$ multiplications ~ 800
3) Backsolve and conversion	$\sim n \cdot (m+2) \cdot k^2$ multiplications ~ 1536

Table 5.1: Work estimates for the three major spline least squares processes with numerical values computed for the "Splines" curve.

The computation of the normal equations need not be postponed until the last data point has been acquired. As each data point is received, the nonzero terms of the sums

(3.5-6) to which that point contributes may be computed and added to the appropriate elements of the least squares arrays. Unless the data point is needed for display or for later comparison with the fit, it need not be stored at all. A data point satisfying

(5.1) $i \cdot h \leq t < (i+1) \cdot h$

affects only rows i+1 to i+k of $\tilde{B}$ and of the lower triangle of G (see Figure 4.1); consequently, after the last point of interval i has been acquired, the values of rows 1 to i+1 are final and are available for further processing.

As each row of G is determined, its factorization may be computed from (4.16-17). The elements ℓ_{ij} and d_{ii} of the factorization are computed in left-to-right order, and for efficient storage utilization they replace the corresponding elements of G. After the factorization is complete to row i, the forward solve may be completed up to element w_i and the elements of W may replace the corresponding elements of B (see Figure 5.2).

The third process -- backsolve and conversion -- cannot be begun until both the forward solution and the factorization are complete. Neither the forward solution nor the factorization can be completed until all of the data have been received; therefore the algorithm as stated will not allow computation of the value of any basis

G:	d										B:	w
	ℓ	d										w
	ℓ	ℓ	d									w
	ℓ	ℓ	ℓ	d								w
		ℓ	ℓ	ℓ	d							w
			ℓ	ℓ	ℓ	d						w
i+1				g	g	g	g					b
⋮					g	g	g	g				b
						g	g	g	g			b
i+k							g	g	g	g		b

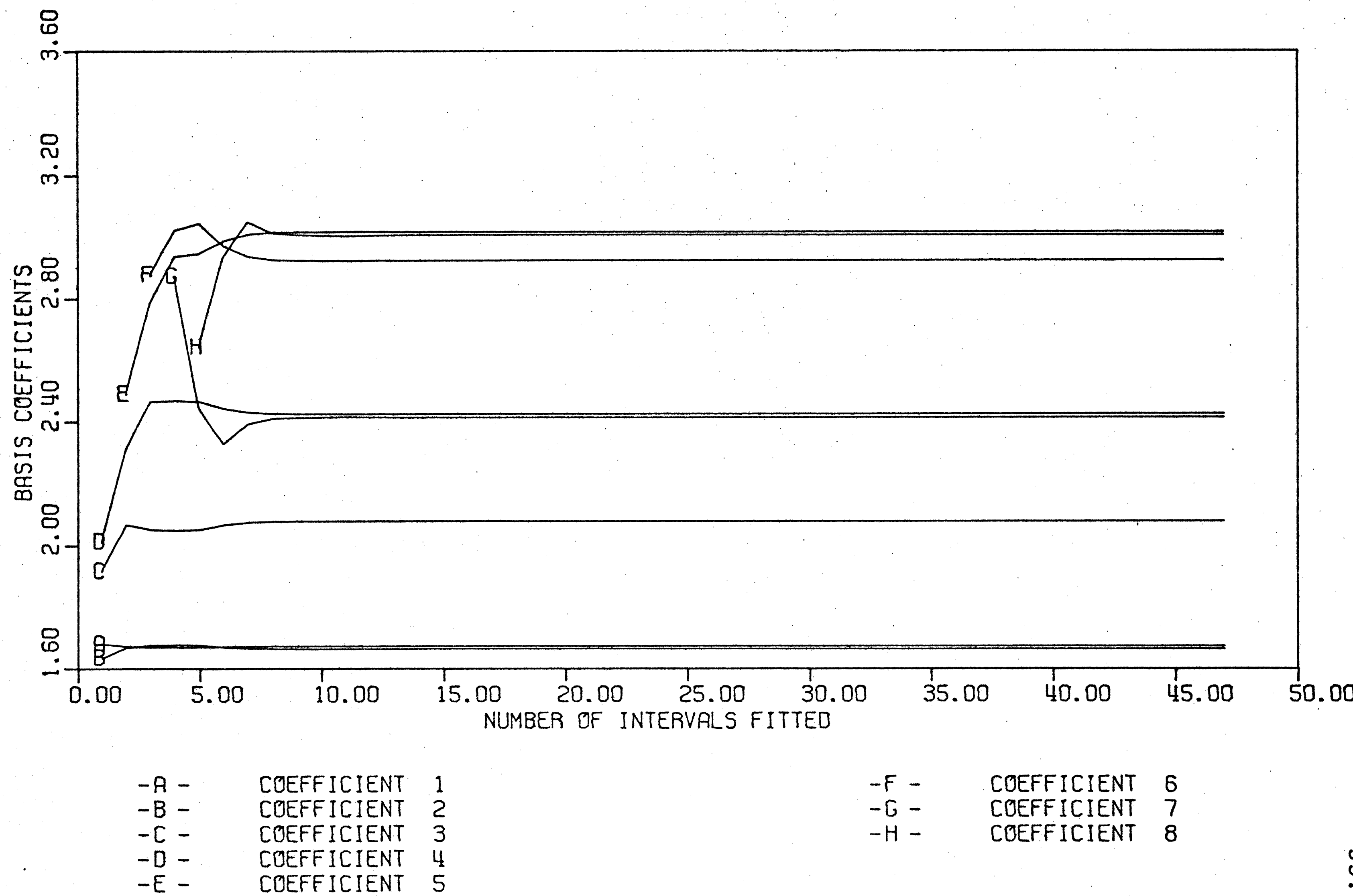
Figure 5.2: The least squares matrices and their factorization during processing of data point t_q satisfying equation 4.1.

coefficient before all of the data have been acquired. Furthermore, since the value of each point on a least squares spline curve depends on the values of all of the data, no method exists for computing any part of a least squares curve without knowing the entire set of data.

However, as the experiment of Figure 5.3 suggests, one may compute approximate values for some of the basis coefficients without the complete set of data. In Figure 5.3, for the "Splines" fit of Figure 5.1, the values of the first eight B-spline basis coefficients were computed using data in the first I intervals and the computed values are plotted for values of I from 1 to 47. As the fraction of data fitted increases, the values of the coefficients approach those computed from the full set of data. For these first eight coefficients, three digit accuracy may be obtained using data in the beginning ten to fifteen intervals only.

This effect is a result of the local convergence property of splines. Stated informally, local convergence implies that the value of a spline at a given point depends mostly on the data in the neighborhood of that point, and that the value of a basis coefficient depends mostly on the data in the neighborhood of the intervals where the corresponding basis function is non-zero. Data far from a given point on a curve affect only slightly the value of

Figure 5.3: Convergence of least squares coefficients to final values as fraction of data fitted increases. Data are from "Splines" curve of Figure 5.1a.



the least squares fit near that point. More precise statements of this property and proofs for some special cases may be found in Powell [14] and Strang and Fix [20]. A proof for the general case appears in Lewis [11].

In interactive graphics applications, where two or three digit accuracy is quite adequate, local convergence allows computation of the spline fit to the first part of curve data before the entire set of data is acquired. Given the first i+1 rows of the forward solve W and of the factorization L\D, the backsolve for rows 1 to i+1 may be performed. For any standard of accuracy, for i sufficiently large, and for some choice of a positive integer lag < i+1, coefficients

(5.2) $A_{i-lag+1}, \ldots, A_i, A_{i+1}$

will not be sufficiently accurate, but the coefficients

(5.3) $A_1, \ldots, A_{i-lag}$

will be sufficiently accurate and may be converted to a piecewise polynomial for display (Figure 5.4). On subsequent backsolves the coefficients of (5.3) need not be computed again.

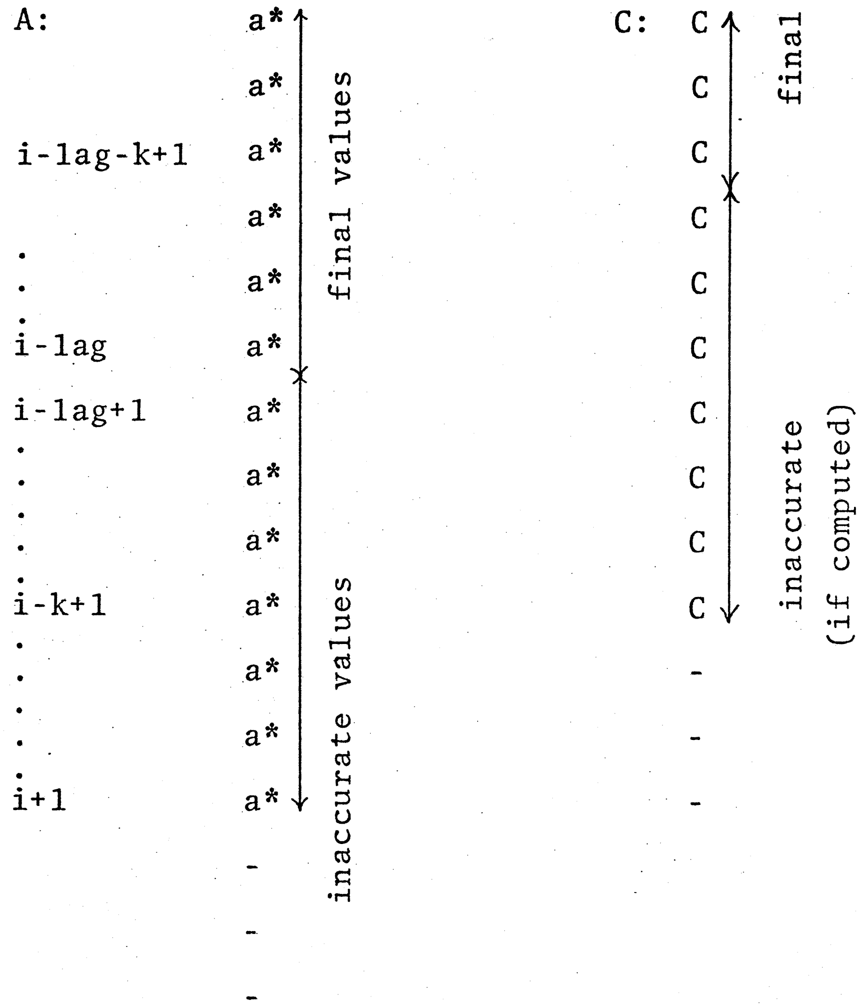


Figure 5.4: Backsolve and conversion.

Assuming that a backsolve is performed at least once per interval and neglecting low order terms, each backsolve requires

(5.4) lag·k multiplications.

Clearly the value of lag should be given the smallest value consistent with working accuracy. For curve data the choice lag = 2·k appears to produce reasonable results without requiring excessive computation.

Other than the additional computation in the backsolve, the computation required for this algorithm is the same as that for the sequential algorithm of Section 4. Assuming that the backsolve is performed once per interval, the computational penalty for obtaining the fit before completion of the curve lies entirely in the backsolve and amounts to approximately

(5.5) [m·lag - (m+k)] · k multiplications

over the sequential algorithm.

If the backsolve is performed once per interval, this scheme requires less storage than the sequential algorithm. After the value for A_{i-lag} has been computed, rows

(5.6) 1, 2, ..., i-lag

of the least squares matrices are no longer required and their storage may be freed for other uses. Only

(5.7) lag • (k+1) locations

of the matrices G and $\underset{\sim}{B}$ need be kept. Therefore, unless the basis coefficients or the piecewise polynomial must be saved, the storage required for the algorithm is _independent of the length of the curve_.

6. Closed Curves

For a general curve-design system, both open and closed curve capabilities are required. Objects such as bowling pins, television tubes, and bottles contain closed cross-section curves, and the designer must be able to create and manipulate these closed curves as easily as open curves. Furthermore, the system to fit and manipulate these closed curves should meet the same goals as the open curve system -- compactness and real-time response.

The entire development of Sections 3 to 5 generalizes to closed curves. The closed spline curve is simply an open curve whose endpoints join smoothly, i.e. an open curve is also a closed curve if it satisfies the periodic end conditions

(6.1) $D^{\ell}S(0^+) = D^{\ell}S(a^-) \quad \text{for } 0 \le \ell \le k-2.$

For $m > k$ this derivative constraint is equivalent to the constraint

(6.2) $\underset{\sim}{A}_i = \underset{\sim}{A}_{i+m+1} \quad \text{for } 1 \le i \le k-1$

on the B-spline basis coefficients.

Except for the additional constraint (6.2), the calculation of the periodic least squares spline curve follows the development of Section 3. Since the curve fitting problem reduces to a number of one variable least squares problems, only the one variable case will be treated here.

Given the order k, the knots

(6.3) Δ: 0, h, ..., m$\cdot$h, (m+1)$\cdot$h = a

and the data

(6.4) $X = \{(t_q, x_q): 1 \le q \le M\}$,

the closed (or periodic) least squares spline is that spline $S_k^\Delta(t)$ which is closest to the data in that it minimizes the distance functional

$$(6.5) \quad d(S,X) = \sum_{q=1}^{M} [S(t_q) - x_q]^2$$

over all splines of given order and knots <u>which satisfy the smoothness constraint</u> (6.2). As in Section 3 the distance functional d(S,X) may be written as a quadratic function of $\underset{\sim}{A}$

$$(6.6) \quad F(\underset{\sim}{A}) = \underset{\sim}{A}^T \cdot G \cdot \underset{\sim}{A} - 2 \cdot \underset{\sim}{A}^T \cdot \underset{\sim}{B} + C$$

where G, $\underset{\sim}{B}$, and C are defined in Section 3.

The constraint (6.2) may be eliminated by substitution of A_{i-m-1} for A_i if $i > (m+1)$ in equation (6.6). After this substitution (6.6) becomes

(6.7) $F(\overset{\circ}{\underset{\sim}{A}}) = \overset{\circ}{\underset{\sim}{A}}^T \cdot \overset{\circ}{G} \cdot \overset{\circ}{\underset{\sim}{A}} - 2 \cdot \overset{\circ}{\underset{\sim}{A}}^T \cdot \underset{\sim}{B} + C$

where the periodic Gram matrix $\overset{\circ}{G}$

$$(6.8) \quad \overset{\circ}{G} = [\overset{\circ}{g}_{ij}] = \begin{bmatrix} g_{ij} + g_{i+m+1,j+m+1} & \text{for } 1 \le j,\ i \le k-1 \\ g_{j+m+1,i} & \text{for } 1 \le j \le k-1 \\ & \text{and } m-k+3 \le i \le m+1 \\ g_{j,i+m+1} & \text{for } 1 \le i \le k-1 \\ & \text{and } m-k+3 \le j \le m+1 \\ g_{i,j} & \text{otherwise} \end{bmatrix}$$

and the periodic vector $\overset{\circ}{\underset{\sim}{B}}$

$$(6.9) \quad \overset{\circ}{\underset{\sim}{B}} = [\overset{\circ}{b}_i] = \begin{bmatrix} b_i + b_{i+m+1} & \text{for } 1 \le i \le k-1 \\ b_i & \text{otherwise} \end{bmatrix}$$

are obtained by "folding" the non-periodic matrices G and $\underset{\sim}{B}$ (Figure 6.1). This "folding" computation requires no multiplications and

(6.10) $\sim\frac{1}{2} k^2$ additions.

The coefficients

$$(6.11) \quad A_i = \left\{ \begin{matrix} \overset{\circ}{A}_i & \text{for } 1 \le i \le m+1 \\ \overset{\circ}{A}_{i-m-1} & \text{for } m+2 \le i \le m+k \end{matrix} \right\}$$

of the corresponding open curve are the periodic extension

	X	X	X	.	.	.	.	.	.	.		X
G:	X	X	X	X	.	.	.	.	.	.	B:	X
	X	X	X	X	X	.	.	.	.	.		X
	.	X	X	X	X	X	.	.	.	.		X
	.	.	X	X	X	X	X	.	.	.		X
	.	.	.	X	X	X	X	X	.	.		X
m-k+3	.	.	.	.	X	X	X	X	1	.		X
m+1	.	.	.	.	.	X	X	X	2	3		X
m+2	.	.	.	.	.	.	1	2	4	5		1
m+k	.	.	.	.	.	.	.	3	5	6		2

Figure 6.1a: The non-periodic matrix G and vector B for $k = 3$, $m = 7$. Nonzero elements are indicated by "X" and zero elements by ".". Numbered elements are the nonzero elements that are added to the non-periodic matrix G to produce the periodic matrix $\mathring{G}$.

$$
\begin{array}{ll}
 & 1 \\
\mathring{G}: & k-1 \\
 & \\
 & \\
 & \\
 & \\
 & m-k+3 \\
 & m+1
\end{array}
\quad
\begin{array}{cccccccc}
4 & 5 & X & . & . & . & 1 & 2 \\
5 & 6 & X & X & . & . & . & 3 \\
X & X & X & X & X & . & . & . \\
. & X & X & X & X & X & . & . \\
. & . & X & X & X & X & X & . \\
. & . & . & X & X & X & X & X \\
1 & . & . & . & X & X & X & X \\
2 & 3 & . & . & . & X & X & X
\end{array}
\qquad
\mathring{B}: \;
\begin{array}{c}
1 \\ 2 \\ X \\ X \\ X \\ X \\ X \\ X
\end{array}
$$

Figure 6.1b: The periodic matrix $\mathring{G}$ and vector $\mathring{B}$ for $k = 3$, $m = 7$. Nonzero elements from the non-periodic matrices are indicated by "X" and zero elements by ".". The numbered elements are the sums of the corresponding elements in the non-periodic matrix and the elements of the same number in Figure 6.1a.

of the closed curve coefficients $\overset{\circ}{\underset{\sim}{A}}{}^*$.

The periodic Gram matrix $\overset{\circ}{G}$ is well conditioned, positive definite, and symmetric; but it is not banded (Figure 6.1b). While band methods are not efficient for such a matrix, variable bandwidth or envelope methods are quite effective [7]. For the symmetric matrix $\overset{\circ}{G}$, the lower half of the envelope of $\overset{\circ}{G}$ consists of the elements

(6.12) $\{\overset{\circ}{g}_{ij}: f_i \leq j \leq i,\ 1 \leq i \leq m+1\}$ where

(6.13) $f_i \equiv \min j$ such that $\overset{\circ}{g}_{ij} \neq 0$

Only the envelope of the matrix is ever stored or computed. The elements of the lower triangle of the envelope of G are atored in a real vector V^G and accessed through the integer vector U^G

(6.14) $\overset{\circ}{g}_{ij} = V^G_{j+u^G_i}$

with

(6.15) $u^G_i = \sum_{j=1}^{i} (j - f_j)$.

Storage of the periodic matrix G requires fewer than

(6.16) $(m+1) \cdot (2k-1)$ locations for V^G

and $(m+1)$ locations for the pointers U^G.

Figure 6.2 illustrates the storage of and access to the envelope of a periodic Gram matrix.

									F:	U^G:
	E								1	0
G:	E	E							1	1
	E	E	E						1	3
		E	E	E					2	5
			E	E	E				3	7
				E	E	E			4	9
	E	E	E	E	E	E	E		1	15
	E	E	E	E	E	E	E	E	1	22

V^G g_{11} $g_{21}g_{22}$ $g_{31}g_{32}g_{33}$ $g_{42}g_{43}g_{44}$ $g_{53}g_{54}g_{55}$ $g_{64}g_{65}g_{66}$

$g_{71}g_{72}g_{73}g_{74}g_{75}g_{76}g_{77}$

$g_{81}g_{82}g_{83}g_{84}g_{85}g_{86}g_{87}g_{88}$

Figure 6.2: The lower half of the envelope of the Gram matrix G for k = 3, m = 7. Elements of the envelope are indicated by "E".

The solution of the normal equations is obtained in much the same manner as in Sections 4 and 5. Since $\mathring{G}$ is positive definite, the factorization

(6.17) $\mathring{G} = L \cdot D \cdot L^T$

exists. From the relations [7]

$$(6.18)\quad \ell_{ij} = (g_{ij} - \sum_{q=\max(f_i,f_j)}^{i-1} d_{qq} \cdot \ell_{iq} \cdot \ell_{jq})/d_{jj}$$

$$\text{for } 1 \le j \le i \le m+k$$

and

$$(6.19)\quad d_{ii} = g_{ii} - \sum_{q=f_i}^{i-1} d_{qq} \cdot \ell_{iq}^2 \qquad \text{for } 1 \le i \le m+k$$

the elements of D and L may be computed. Just as the factorization of G for the non-periodic case has the same bandwidth as G, so the factorization of $\mathring{G}$ has the same envelope as $\mathring{G}$. Consequently, the factorization of $\mathring{G}$ may replace $\mathring{G}$ in memory just as before. After a forward solve

$$(6.20)\quad \mathring{\underset{\sim}{W}} = L^{-1} \cdot \mathring{\underset{\sim}{B}}$$

for $\mathring{\underset{\sim}{W}}$, the solution $\mathring{\underset{\sim}{A}}{}^*$ is obtained from the back solution

$$(6.21)\quad \mathring{\underset{\sim}{A}}{}^* = (L^T)^{-1} \cdot D^{-1} \cdot \mathring{\underset{\sim}{W}}.$$

For an n-dimensional problem and neglecting low order terms, the factorization requires

$$(6.22)\quad 2 \cdot (m+1) \cdot k^2 \text{ multiplications,}$$

the forward solve requires

(6.23) $2 \cdot n \cdot (m+1) \cdot k$ multiplications,

and the back solve requires

(6.24) $2 \cdot n \cdot (m+1) \cdot k$ multiplications.

In practice the curve would be drawn and fit as an open curve (Figure 6.3) and the ends would be joined later. If the designer were so kind as to draw a curve whose length was an integer multiple of the knot spacing h, i.e.

(6.25) $a = h \cdot (m+1)$

then the fit could be obtained easily by "folding" the non-periodic matrices and solving the resulting linear system (Figure 6.4). Unfortunately, if the designer were to draw a curve not satisfying equation (6.25), kinks and bulges could be expected if the closed curve were calculated from the open curve fit (Figure 6.5) by folding (Figure 6.6).

If equation (6.25) is not satisfied, to obtain an exact periodic least squares fit the entire computation could be repeated with

(6.26) $h^{new} = \frac{a}{m+1}$.

This fit would require considerable computation -- more than the original open curve fit itself -- and might introduce unacceptable delay in an interactive system. Furthermore,

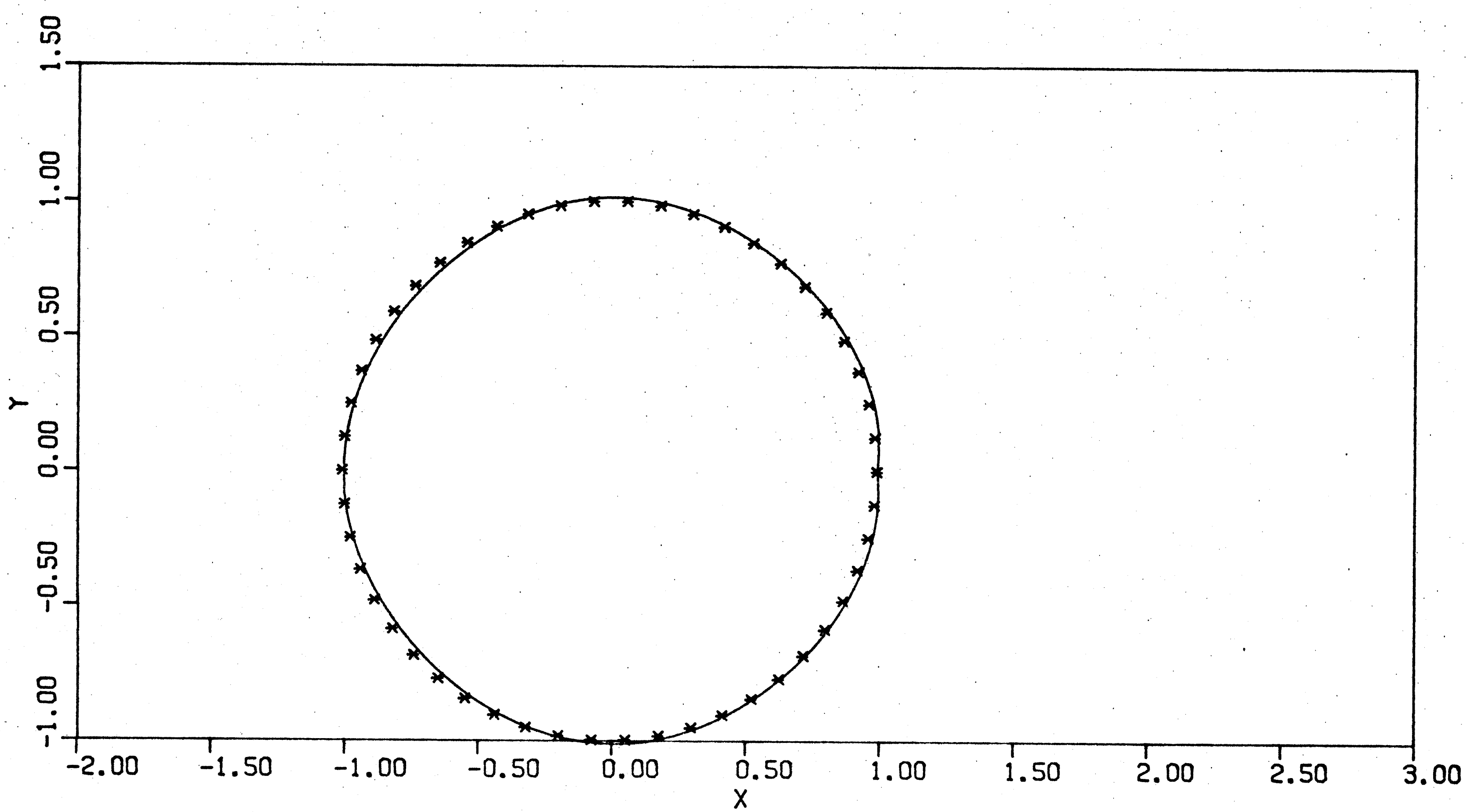


Figure 6.3: Open curve fit, $5 \cdot h = a$, $m = 4$, $k = 4$.

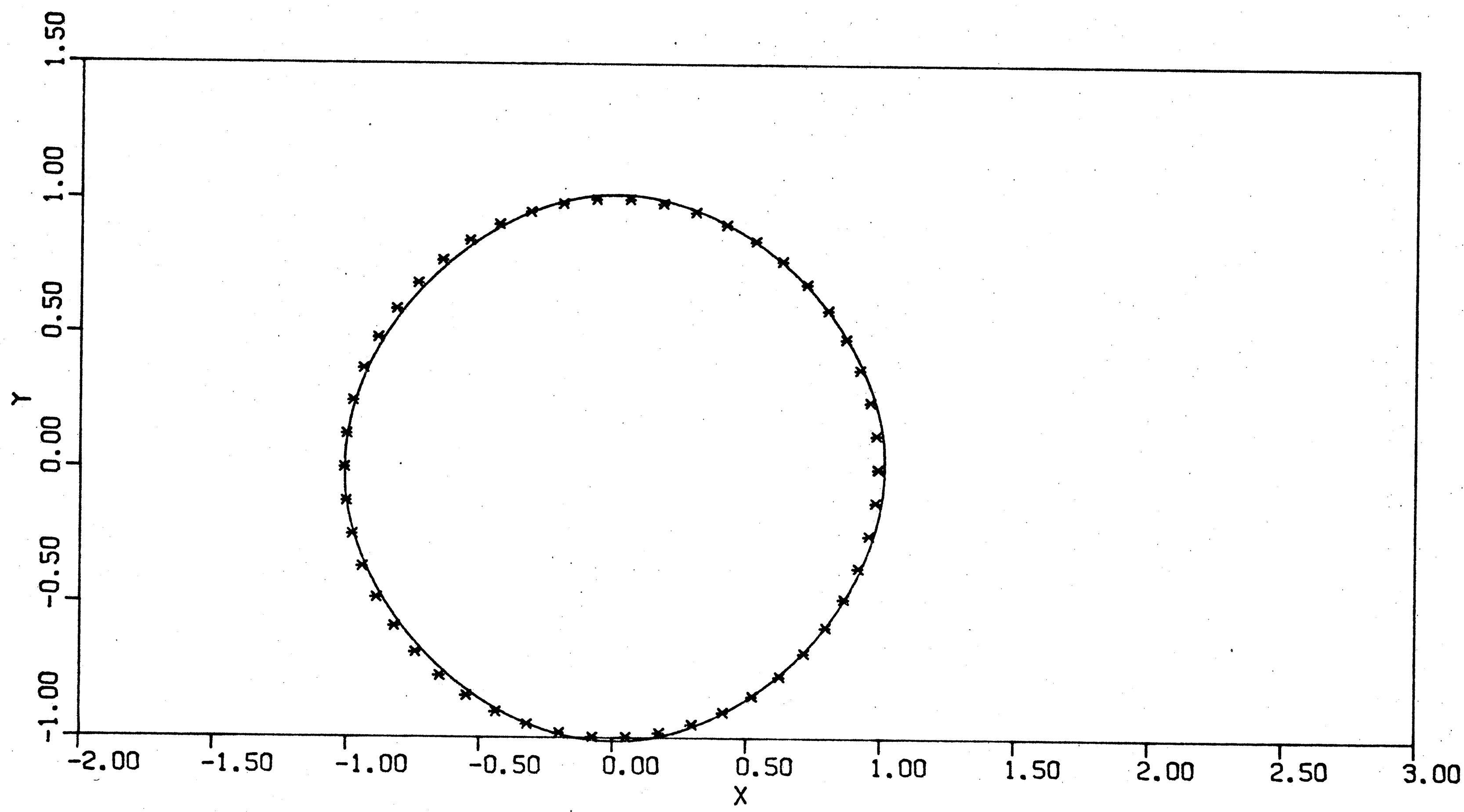


Figure 6.4: Closed curve fit, 5•h = a, m = 4, k = 4.

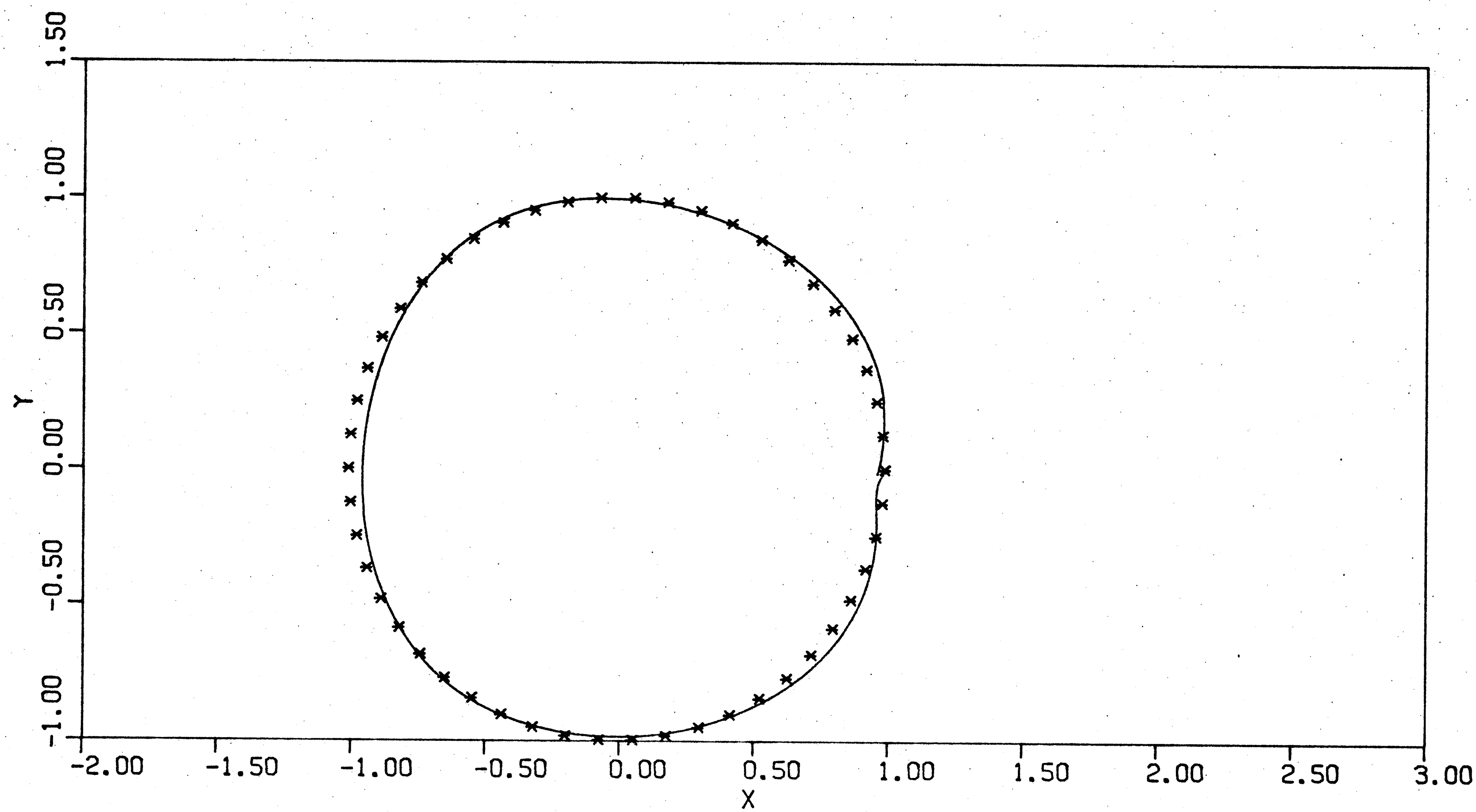


Figure 6.5: Open curve fit, $4.1 \cdot h = a$, $m = 4$, $k = 4$.

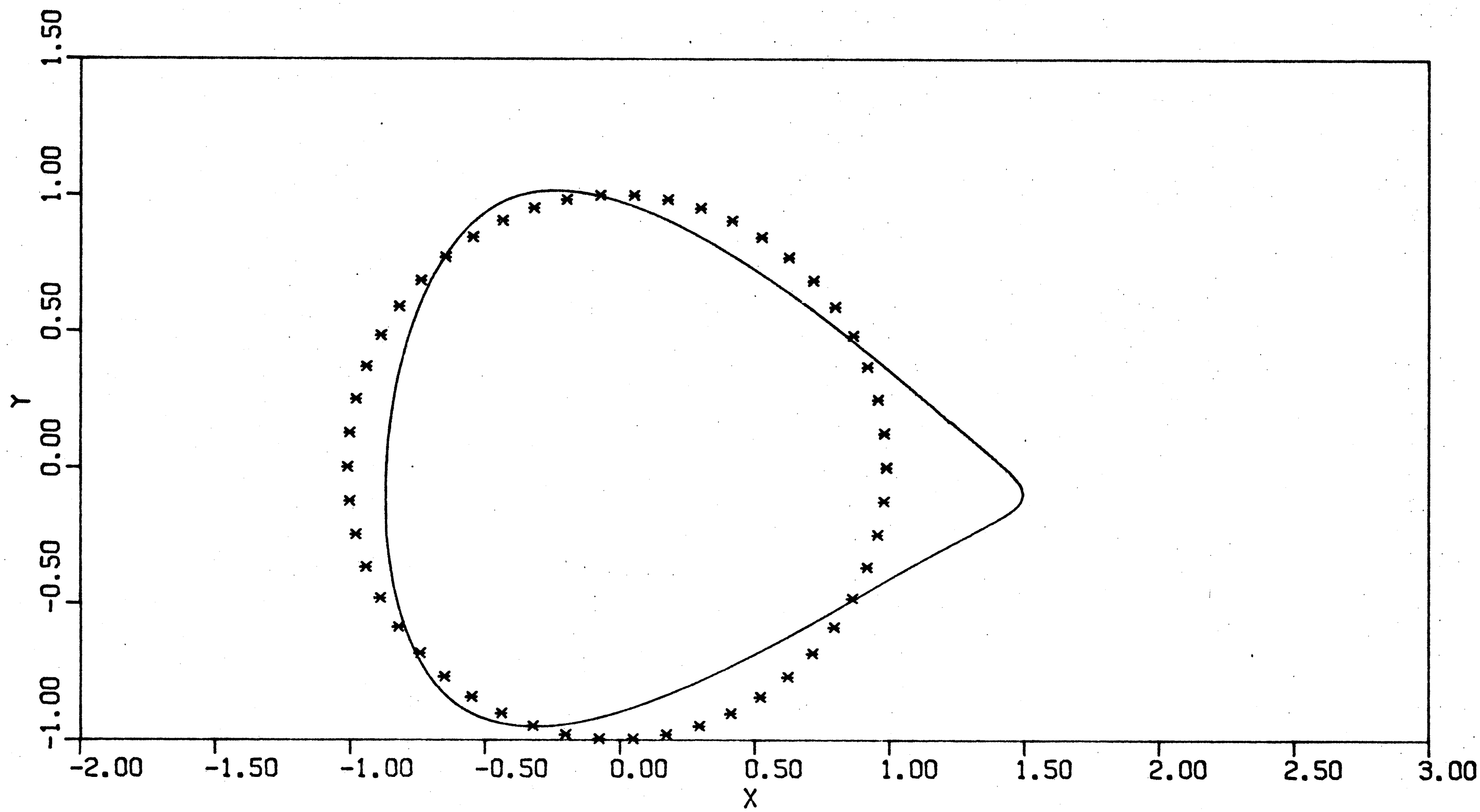


Figure 6.6: Closed curve fit, 4.1·h = a, m = 4, k = 4.

all of the data would have to be stored. To avoid much of this delay and the requirement of data storage, an approximate closed curve fit can be obtained by fitting a closed curve to points evaluated from the open curve fit (Figure 6.5 illustrates the open curve fit and Figure 6.7 is a closed curve fit to data from that fit). If the evaluation points are chosen carefully, such as at the Gaussian quadrature points, relatively few evaluations are necessary and the time required for a closed curve fit is a small fraction of the computation time for the exact fitting procedure.

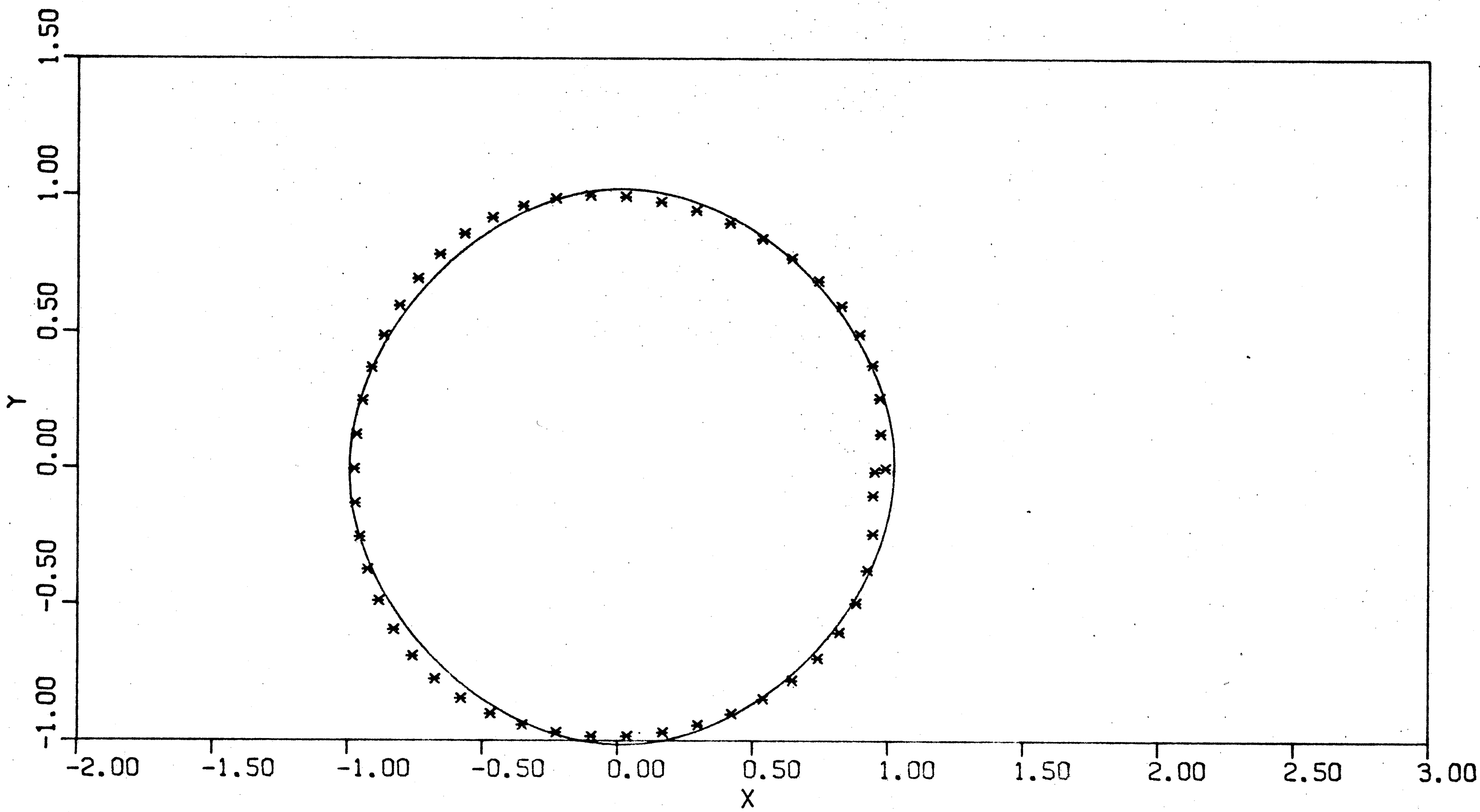


Figure 6.7: Closed curve fit using data obtained from fit of Figure 6.5, 5·h = a, m = 4, k = 4.

7. Acknowledgments

Thanks are due Barbara Pick of Haskins Laboratories for the use of a graphics tablet and her penmanship in the "Splines" curve data.

References

1. C. de Boor.
On calculating with B-splines.
JAT 1, 50-68, 1972.

2. C. de Boor.
The B-spline package.
SIAM Journal on Numerical Analysis, to appear.

3. M. G. Cox.
The numerical evaluation of B-splines.
Report DNAC 4, National Physical Laboratory, Teddington, England, August 1971.

4. H. B. Curry and I. J. Schoenberg.
On Polya frequency functions IV: the fundamental spline functions and their limits.
J. Analyse Math. 17, 71-107, 1966.

5. S. C. Eisenstat.
L^∞-convergence of discrete least squares splines.
Yale Computer Science Research Report.

6. S. C. Eisenstat, J. W. Lewis, and M. H. Schultz.
FITS: a subroutine package for spline regression.
Yale Computer Science Research Report.

7. S. C. Eisenstat and A. H. Sherman.
Subroutines for envelope solution of sparse linear systems.
Yale Computer Science Research Report #35.

8. A. R. Forrest.
Interactive interpolation and approximation by Bezier polynomials.
Comp J. 15, 71-79, 1972.

9. G. Forsythe and C. B. Moler.
Computer Solution of Linear Algebraic Systems.
Prentice Hall, 1967.

10. W. J. Gordon and R. F. Riesenfeld.
Berstein-Bezier methods for the computer-aided design of free-form curves and surfaces.
JACM 21, 293-310, 1974.

11. J. W. Lewis.
Data Fitting and Constrained Spline Regression.
Doctoral dissertation, Yale University, to appear.

12. J. W. Lewis.
Spline curve editing.
Yale Computer Science Research Report.

13. R. S. Martin and J. H. Wilkinson.
Symmetric decomposition of positive definite band matrices.
Num. Math 7, 355-361, 1965.

14. M. J. D. Powell.
The local dependence of least squares cubic splines.
SIAM Journal on Numerical Analysis 6, 398-413, 1969.

15. R. Riesenfeld.
Applications of B-spline approximation to geometric problems of computer aided design.
Report UTEC-CSc-73-126, Department of Computer Science, University of Utah.

16. M. H. Schultz.
L^2-convergence of discrete least squares splines.
Yale Computer Science Research Report.

17. M. H. Schultz.
Spline Analysis.
Prentice Hall, 1973.

18. M. H. Schultz and R. S. Varga.
L-splines.
Numer. Math. 10, 345-369, 1967.

19. G. Strang and G. Fix.
A Fourier analysis of the finite element method.
Proceedings of the CIME Summer School, Italy.

20. R. S. Varga.
Functional analysis and approximation theory in numerical analysis.
SIAM Regional Conference Series in Applied Mathematics #3, Philadelphia, 1971.

Appendix I: FORTRAN IV Codes

The algorithm of this paper has been implemented in FORTRAN IV and tested with the PDP-10 F4 compiler, the OS370 G compiler, and the CDC RUN compiler. The codes are close to the ASA standard for FORTRAN but do violate the standards for integer expressions in array indices. There are two basic packages -- CURVS, the curve fitting package, and SPLSQ, a package for weighted least squares splines in one dimension. Both packages use the modules in PSQARS (primitive least squares operations) and ENSOLV (incremental envelope linear system solution).

All of the code is well-documented and highly modular. The code was designed more to illustrate the implementation of the algorithm than for efficiency. For maximum efficiency, the subroutine parameters should be placed in COMMON blocks and some of the subroutine code should be placed in-line.

Use of the packages is illustrated by a set of drivers. All drivers include sample data files and part of the resulting output so that the proper functioning of the codes may be verified conveniently. There are three drivers for the CURVS package, each illustrating a different mode of use. These three modes are explained at the beginning of the CURVS module. There is one simple data fitting driver for the SPLSQ subroutine.

The SPLSQ subroutine is well-protected against user error in that it checks subroutine parameters and in case of error returns an explanatory flag; but because of its structure, the CURVS package could not be so well-protected. Variables and arrays passed from subroutine to subroutine (for example, JG, G, LOW, LOWF) should not be modified, and the subroutines must be called in the specified order.

Machine-readable copies of the codes are available from

Numerical Software
Department of Computer Science
Yale University
New Haven, Connecticut 06520.

```
C     ***************************************************************
C     *                    Module SPLSQ                             *
C     *            Least Squares Polynomial Splines                 *
C     *                  User Subroutine                            *
C     *                     22 Sep 74                               *
C     ***************************************************************
C     Entries:   SPLSQ
C     Externals: from module PSQARS: SETUPS,ADDON,FOLD,CONVRT
C                from module ENSOLV: FACTOR,SOLVE
C
C     Reference:   S.C. Eisenstat, J.W. Lewis, M.H. Schultz,
C     "A Real-Time Algorithm for Least Squares Splines and
C     Its Application in Computer Aided Geometric Design",
C     research report #29, Department of Computer Science,
C     Yale University
C
C        Subroutine SPLSQ fits a periodic or non-periodic least squares
C spline to weighted data  XD(ND), WD(ND), Y(LY,NY).  The least squares
C spline is returned as the basis coefficients A(LA,NY) and as the
C piecewise polynomial CPP(LCK,LCM,NY).  The subroutine is well
C protected against user error.  If parameters are incorrect, SPLSQ
C returns with an explanatory error flag.
C
C***************Subroutines and Calling Sequences*********************
C
C 1) Unless otherwise indicated, variable types are
C        IMPLICIT REAL(A-H, O-Z)
C        IMPLICIT INTEGER(I-N)
C 2) Value variables (V) pass a value to the subroutine and must
C    have been set by the user or by a previously called subroutine.
C    Result variables (R) return results to the calling subprogram.
C
C       CALL SPLSQ( JPER, K, MK,AA,BB, ND,XD,WD, NY,LY,YD,
C    1  JG,LG,G,NG, LA,A, LCK,LCM,CPP,HKNOT, LFLAG)
C
C V  JPER          - Nonzero to indicate periodic problem
C V  K             - Degree +1 of spline
C V  MK            - Number of knots (including endpoints)
C V  AA            - Left endpoint of interval
C V  BB            - Right endpoint of interval
C
C V  ND            - Number of evaluation points
C V  XD(ND)        - Evaluation points
C V  WD(ND)        - Weight for each evaluation point
C                    if WD(1)<0 then points are equally weighted
C V  NY            - Number of sets of ordinate values
C V  LY            - Dimension  of YD
C V  YD(LY,NY)     - NY sets of ND ordinate values
C
C R  JG(LA)        - Index array for grammian
C V  LG            - Dimension of G
C R  G(LG)         - Storage for grammian (profile only is stored)
C                    G(I,J) = G( JG(I) + J )    for   I le J
C R  NG            - Number of locations used in G
C
C V  LA          - Dimension  of A
C R  A(LA,NY)    - Basis coefficients
C V  LCK,LCM     - Dimensions of CPP
C R  CPP(LCK,LCM,NY) - Piecewise polynomial
C R  HKNOT       - Knot spacing  (for use by EVAL)
C
C R  LFLAG       - Status (either -1, 0 or a sum of the positive codes)
C
C                   -1  - Problem is ill posed - badly distributed data
C                    0  - Successful fit
C                    1  - K  LT 1
C                    2  - K GT KMAX  or K GT LCK
C                    4  - MK LT 2
C                    8  - (MK-1) GT LCM
C                   16  - MK LE K  and  JPER ne 0
C                   32  - AA LE BB
C                   64  - NY LT 1
C                  128  - NA GT LA
C                  256  - NG GT LG
C                  512  - N  GT ND    insufficient data
C
C***SPLSQ*************************************************************
      SUBROUTINE SPLSQ( JPER, K, MK,AA,BB, ND,XD,WD, NY,LY,YD,
     1  JG,LG,G,NG, LA,A, LCK,LCM,CPP,HKNOT, LFLAG)
       DIMENSION  XD(NY), WD(NY), YD(LY,NY)
       DIMENSION  JG(LA),G(LG), A(LA,NY), CPP(LCK,LCM,NY)
       DIMENSION  CB(36)
       DATA KMAX/6/
C
C  Compute storage used and check parameters
C
       NA = MK+K-2
       N = NA
       IF(JPER.NE.0) N = MK-1
       NG = NA*K - K*(K-1)/2
C  Error checks
       LFLAG = 0
       IF( K.LE.0                     )          LFLAG = LFLAG +   1
       IF((K.GT.KMAX).OR.(K.GT.LCK) )            LFLAG = LFLAG +   2
       IF( MK.LT.2                    )          LFLAG = LFLAG +   4
       IF( (MK-1).GT.LCM              )          LFLAG = LFLAG +   8
C  Additional restriction for periodic problems
       IF(JPER.EQ.0) GO TO 20
         IF( MK.LE.K                  )          LFLAG = LFLAG +  16
         KM1 = K-1
         JF = MK-K
         DO 10 I=1,KM1
10          NG = NG + MAX0( 0, I+JF -K )
20     IF( AA .GE. BB                 )          LFLAG = LFLAG +  32
       IF( NY .LE. 0                  )          LFLAG = LFLAG +  64
       IF( NA .GT. LA                 )          LFLAG = LFLAG + 128
       IF( NG .GT. LG                 )          LFLAG = LFLAG + 256
       IF( N  .GT. ND                 )          LFLAG = LFLAG + 512
       IF( LFLAG.NE.0 ) RETURN
C  Setups
```

```
C     ***************************************************************
C     *                       SPLSQ driver                          *
C     *              Least Squares Polynomial Splines               *
C     *                        24 Aug 74                            *
C     ***************************************************************
C     Externals: from module SPLSQ:  SPLSQ
C                from module PSQARS: EVAL
C
C     Reference:   S.C. Eisenstat, J.W. Lewis, M.H. Schultz,
C     "A Real-Time Algorithm for Least Squares Splines and
C     Its Application in Computer Aided Geometric Design",
C     research report #29, Department of Computer Science,
C     Yale University
C
C     This program is a driver for the SPLSQ spline fitting subroutine.
C     It reads data from a file and prints the piecewise polynomial
C     for the fit.
C
C     For LFLAG error messages, see module SPLSQ
C
C Sample data file (sine and cosine)
C...04 5 112                            JPER,K,MK,ND,NY  (2I1,I2,I2,I1)
C...   0.000000  6.283185               Interval of fit  (2F10.6)
C...   0.000000  0.000000  1.000000     Data   XD,  YD(1), ....,YD(NY)
C...   0.628319  0.587785  0.809017            (8F10.6)
C...   1.256637  0.951057  0.309017
C...   1.884956  0.951057 -0.309017       JPER - end condtions
C...   2.513274  0.587785 -0.809017              periodic for JCON NE 0
C...   3.141593  0.000000 -1.000000       K    - Degree + 1
C...   3.769911 -0.587785 -0.809017       ND   - Number of data points XD
C...   4.398230 -0.951057 -0.309017       NY   - Number of YD values
C...   5.026548 -0.951057  0.309017              at each point XD
C...   5.654867 -0.587785  0.809017
C...   6.283185 -0.000000  1.000000
C
C Results:
C
C         SPLINE FIT
C   JPER  K  MK  ND  NY      AA         BB
C     0   4   5  11   2    0.0000     6.2832
C   PIECEWISE POLYNOMIAL... 1
C        0.00124      0.98717     -0.02238     -0.12471
C        1.01332     -0.00626     -0.61005      0.12946
C        0.00000     -0.96452      0.00000      0.12946
C       -1.01332     -0.00626      0.61005     -0.12471
C   PIECEWISE POLYNOMIAL... 2
C        0.99916      0.08489     -0.70341      0.15461
C       -0.00385     -0.98048      0.02517      0.12177
C       -1.00990     -0.00000      0.59902     -0.12177
C       -0.00385      0.98048      0.02517     -0.15461
C
      DIMENSION XD(101), WD(1), YD(120,6)
      DATA  NDMAX/101/, LY/120/, NYMAX/6/
      DIMENSION JG(20),  A(20,6), G(300), CPP(6,20,6)
      DATA  LA/20/, LG/300/, LCK/6/, LCM/20/
C     Uniform weighting of data
        WD(1) = -1.
C
C      Read Data
C
  100 READ(5,11010)  JPER,K, MK, ND, NY, AA, BB
11010 FORMAT(2I1,I2,I3,I1/2F10.4)
      WRITE(6,11020) JPER,K,MK,ND,NY,AA,BB
11020 FORMAT(//'          Spline Fit '/
     1  ' JPER  K  MK  ND  NY     AA        BB  '/
     2   5I4,2F10.4)
      IF( (ND.LE.NDMAX).AND.(NY.LE.NYMAX) ) GO TO 110
        WRITE(6,11030)
11030   FORMAT(' Bad input parameters')
        STOP
  110 DO 120 I=1,ND
  120   READ(5,11040) XD(I), (YD(I,J), J=1,NY)
11040 FORMAT(8F10.6)
C
C     Fit data
C
  200 CALL SPLSQ( JPER, K, MK,AA,BB, ND,XD,WD, NY,LY,YD,
     1   JG,LG,G,NG, LA,A, LCK,LCM,CPP,HMESH, LFLAG )
C     Check for error return
      IF(LFLAG.EQ.0) GO TO 210
        WRITE(6,12010)  LFLAG
12010   FORMAT(' ERROR   ',I10)
        STOP
  210 CONTINUE
C
C     Print Piecewise Polynomials
C
  300 MKM1 = MK-1
      DO 320 L=1,NY
        WRITE(6,13010) L
13010   FORMAT(/' Piecewise polynomial...',I2)
        DO 310 M=1,MKM1
  310     WRITE(6,13020) (CPP(J,M,L), J=1,K)
13020   FORMAT(6F13.5)
  320   CONTINUE
      STOP
      END
```

```
     1   ', HKNOT = ',F8.4// ' DATA... '/)
       CALL CURV1( K,CB,HKNOT, JG,G,  NY,LA,A, LOW,LOWF,JS, T )
C
C   JCON may be either 0 or 1
C        = 0 Factorization is computed after data are read
C          1 Factorization is computed as the data are read
C
       JCON = 0
       DO 10 I=1,ND
         READ(5,10030) Y
         WRITE(6,10030) Y
10030    FORMAT(2F10.6)
         CALL CURV2( JCON, K,CB,HKNOT, LOW,LOWF,T, Y,YO,
     1     JG,G,G, NY,LA,A,A, LFLAG )
         IF(LFLAG.NE.0) GO TO 1001
   10    JD = I
C   Compute fit
   20  CALL CURV3( JCON, K,CB, LA, LOW,LOWF,JS, JG,G,G,
     1     NY,LA,A, A,A, LCK,LCM,CPP, MK,  LFLAG )
       IF(LFLAG.NE.0) GO TO 1001
C
C   Evaluate fit
C
       WRITE(6,10040)
10040  FORMAT(//'  Fit evaluation .. '/)
       DO 30 I=1,NE
         TX = (I-1)*T/(NE-1)
C   Compute fit
         E1 = EVAL( K,MK,0.,HKNOT,LCK,CPP(1,1,1), TX )
         E2 = EVAL( K,MK,0.,HKNOT,LCK,CPP(1,1,2), TX )
   30    WRITE(6,10030) E1,E2
C   Print out K, MK, HKNOT, T, and  basis coefficients
       NB = MK+K-2
       WRITE(6,10050) K,MK,HKNOT, T, NB, (A(I,1),A(I,2),I=1,NB)
10050  FORMAT(//'1 Spline fit  K = ',I2,', MK = ',I3,', HKNOT = ',F10.6
     1  /'     Total arc length = ',F10.6//
     2  1X,I3,' Basis coefficients...'/(2F10.6))
C   Print out piecewise polynomial
       MKM1 = MK-1
       DO 40 L=1,NY
         WRITE(6,10060)
10060    FORMAT(/' Piecewise polynomial... ')
         DO 40 I=1,MKM1
   40      WRITE(6,10070) (CPP(J,I,L), J=1,K)
10070  FORMAT(8F10.6)
       STOP
C   ERROR
 1001  WRITE(6,10080)  LFLAG, JD, LOW, JS, T
10080  FORMAT(' ERROR   ',I10/' JD,LOW,JS ',3I5,', T = ',F8.2)
       STOP
       END
```

```
C   ***************************************************************
C   *                        CURVS driver                          *
C   *            Incremental Least Squares Splines                 *
C   *                         Mode 2 Test                          *
C   *                          29 Aug 74                           *
C   ***************************************************************
C   Externals: from module CURVS:  CURV1,CURV2,CURV3
C              from module PSQARS: EVAL
C
C   Reference:   S.C. Eisenstat, J.W. Lewis, M.H. Schultz,
C   "A Real-Time Algorithm for Least Squares Splines and
C   Its Application in Computer Aided Geometric Design",
C   research report #29, Department of Computer Science,
C   Yale University
C
C   This program is a driver for the CURVS incremental curve
C   fitting package.   It reads data from a file and prints the
C   basis coefficients and piecewise polynomial for the fit.
C
C   For LFLAG error messages, see module CURVS
C
C A sample data file:
C...04   10   50   6.5        K,ND,NE,HKNOT  (I2,2I5,F10.3)
C...  1.         9.           X,Y          (2F10.6)
C...  2.         8.5
C...  3.         8.           K     is the degree+1  (K le 6)
C...  4.         6.5          ND    is the number of data points
C...  4.5        5.           NE    is the number of evaluation points
C...  5.         3.           HKNOT is the knot spacing
C...  4.         2.5
C...  3.         3.
C...  2.         4.
C...  1.         5.
C
C    Resulting Piecewise polynomials (to check operation of program)
C
C     1.071615  0.733437  0.051666 -0.011503    For X coordinate
C     4.862930 -0.052865 -0.172635  0.012959
C
C     8.858785  0.167781 -0.279348  0.021057    For Y coordinate
C     3.929716 -0.794753  0.131266  0.004390
C
       DIMENSION Y(2), YO(2)
       DATA NY/2/
       DIMENSION A(80,2), B(80,2), JG(80), G(6,80)
       DIMENSION CPP(6,80,2), CB(6,6)
       DATA  LCK/6/, LA/80/, LCM/80/
C
C    Fit with Spline
C
C    Read K,ND,HKNOT and form matrices
       READ(5,10010) K, ND,NE, HKNOT
10010  FORMAT(I2,2I5,F10.3)
       WRITE(6,10020) K, ND,NE, HKNOT
10020  FORMAT(//'1  CURVS test  K =',I2,', ND = ',I5,', NE = ',I5,
```

```
      HKNOT = (BB - AA)/(MK-1)
      CALL SETUPS( JPER, K,CB, HKNOT,NA, JG,G, NY,LA,A )
C
C  Form normal equations
C
      XL = AA
      LOW = 0
      WW = 1.
      DO 40 L=1,ND
        IF( WD(1).GE.0 ) WW = WD(L)
        DX = XD(L) - XL
        IF( (DX.GT.0).AND.(DX.LT.HKNOT) ) GO TO 30
C  Compute new LOW and HKNOT
          LOW = MAX0( 0, MIN0( MK-2, INT( (XD(L)-AA)/HKNOT )))
          XL  = AA + LOW*HKNOT
          DX  = XD(L)-XL
C  Add into matrices
   30 CALL ADDON( K,CB, DX,WW, NY,LY,YD(L,1), LOW, JG,G, LA,A )
   40 CONTINUE
C  Fold for periodics
      IF( JPER.NE.0 ) CALL FOLD( K,MK, JG,G, NY,LA,A )
C
C  Solve linear systems
C
      CALL FACTOR( 1,N, JG,G,G, NY,LA,A,A, LFLAG )
C  Error return if system not positive definite
      IF(LFLAG.NE.0) RETURN
      CALL SOLVE( 1,N, JG,G, NY,LA,A,A )
C  Periodic extension of coefficients
      IF(JPER.EQ.0) GO TO 60
      IF(KM1.LE.0) GO TO 60
        DO 50 L=1,NY
          DO 50 I=1,KM1
   50       A(I+MK-1,L) = A(I,L)
C
C  Convert from B-spline to Piecewise polynomial
C
   60 CALL CONVRT( K,CB, 1,MK, NY,LA,A, LCK,LCM, CPP )
      RETURN
      END
C****END****SPLSQ*************************************************************
```

```
C   ****************************************************************
C   *                      CURVS driver                            *
C   *            Incremental Least Squares Splines                 *
C   *                      Mode 1 Test                             *
C   *                      29 Aug 74                               *
C   ****************************************************************
C   Externals: from module CURVS:  CURV1,CURV2,CURV3
C              from module PSQARS: EVAL
C
C   Reference:   S.C. Eisenstat, J.W. Lewis, M.H. Schultz,
C   "A Real-Time Algorithm for Least Squares Splines and
C   Its Application in Computer Aided Geometric Design",
C   research report #29, Department of Computer Science,
C   Yale University
C
C   This program is a driver for the CURVS incremental curve
C   fitting package.   It reads data from a file and prints the
C   basis coefficients and piecewise polynomial for the fit.
C
C   For LFLAG error messages, see module CURVS
C
C A sample data file:
C...04   10   50   6.5       K,ND,NE,HKNOT  (I2,2I5,F10.3)
C...  1.        9.           X,Y          (2F10.6)
C...  2.        8.5
C...  3.        8.           K      is the degree+1  (K le 6)
C...  4.        6.5          ND     is the number of data points
C...  4.5       5.           NE     is the number of evaluation points
C...  5.        3.           HKNOT  is the knot spacing
C...  4.        2.5
C...  3.        3.
C...  2.        4.
C...  1.        5.
C
C   Resulting Piecewise polynomials (to check operation of program)
C
C     1.071615  0.733437  0.051666 -0.011503    For X coordinate
C     4.862930 -0.052865 -0.172635  0.012959
C
C     8.858785  0.167781 -0.279348  0.021057    For Y coordinate
C     3.929716 -0.794753  0.131266  0.004390
C
      DIMENSION Y(2), YO(2)
      DATA NY/2/
      DIMENSION A(80,2), JG(80), G(6,80)
      DIMENSION CPP(6,80,2), CB(6,6)
      DATA  LCK/6/, LA/80/, LCM/80/
C
C   Fit with Spline
C
C   Read K,ND,HKNOT and form matrices
      READ(5,10010) K, ND,NE, HKNOT
10010 FORMAT(I2,2I5,F10.3)
      WRITE(6,10020) K, ND,NE, HKNOT
10020 FORMAT(//'1  CURVS test  K =',I2,', ND = ',I5,', NE = ',I5,
```

```
     1   ', HKNOT = ',F8.4//)
C   Initialization
        CALL CURV1( K,CB,HKNOT, JG,G,  NY,LA,B, LOW,LOWF,JS, T )
C
        DO 30 I=1,ND
          JD = I
          READ(5,10030) Y(1),Y(2)
10030     FORMAT(2F10.6)
C   Add contribution to matrices
          CALL CURV2( -1, K,CB,HKNOT, LOW,LOWF,T, Y,YO, JG,G,G,
     1      NY,LA,B,B, LFLAG )
          IF(LFLAG.NE.0) GO TO 1001
          IF(I.LT.K) GO TO 30
C   Compute fit
            CALL CURV3( -1, K,CB, LA, LOW,LOWF,JS, JG,G,G,
     1        NY,LA,B,B,A, LCK,LCM,CPP, MK,  LFLAG )
            IF(LFLAG.NE.0) GO TO 1001
C   Print results
            MKM1 = MK-1
            DO 20 L=1,NY
              WRITE(6,10035) L, (A(J,L), J=1,JS)
10035         FORMAT(/' BASIS COEFFICIENTS...',I2/(6F12.5))
              WRITE(6,10040) L
10040         FORMAT(/' Piecewise polynomial...',I2)
              DO 20 M=1,MKM1
   20           WRITE(6,10050) (CPP(J,M,L), J=1,K)
10050       FORMAT(6F12.5)
   30     CONTINUE
        STOP
C   ERROR
 1001   WRITE(6,10060)  LFLAG, JD, LOW, JS, T
10060   FORMAT(' ERROR   ',I10/' JD,LOW,JS ',3I5,', T = ',F8.2)
        STOP
        END
```

```
C   ****************************************************************
C   *                       CURVS driver                           *
C   *              Incremental Least Squares Splines               *
C   *                        Mode 3 Test                           *
C   *                        29 Aug 74                             *
C   ****************************************************************
C   Externals: from module CURVS:  CURV1,CURV2,CURV3,CURV4
C              from module PSQARS: EVAL
C
C   Reference:   S.C. Eisenstat, J.W. Lewis, M.H. Schultz,
C   "A Real-Time Algorithm for Least Squares Splines and
C   Its Application in Computer Aided Geometric Design",
C   research report #29, Department of Computer Science,
C   Yale University
C
C   This program is a driver for the CURVS incremental curve
C   fitting package.   It reads data from a file and prints the
C   basis coefficients and piecewise polynomial for the fit.
C
C   For LFLAG error messages, see module CURVS
C
C A sample data file:
C... 4   11   25  1.2567         K,ND,NE,HKNOT  (I2,2I5,F10.3)
C...  1.000000  0.              X,Y          (2F10.6)
C...  0.809017  0.587785
C...  0.309017  0.951057
C... -0.309017  0.951057         K     is the degree+1  (K LE 6)
C... -0.809017  0.587785         ND    is the number of data points
C... -1.000000  0.               NE    is the number of evaluation point
C... -0.809017 -0.587785         HKNOT is the knot spacing
C... -0.309017 -0.951057
C...  0.309017 -0.951057
C...  0.809017 -0.587785
C...  1.000000  0.
C
C  Resulting basis coefficients (to check operation of program)
C
C   BASIS COEFFICIENTS... 1
C    0.28908     1.33392     0.37554    -1.09604    -1.01319     0.50419
C    1.33095     0.11767
C
C   BASIS COEFFICIENTS... 2
C   -1.29037     0.00864     1.25699     0.73147    -0.84184    -1.21183
C    0.13368     1.29697
C
      DIMENSION Y(2), YO(2)
      DATA NY/2/
      DIMENSION A(80,2), ALIB(80,2), B(80,2)
      DIMENSION JG(80), G(11,80), GLD(6,80)
      DIMENSION CPP(6,80,2), CB(6,6)
      DATA  LCK/6/, LA/80/, LCM/80/
C
C   Fit with Spline
C
C   Read K,ND,HKNOT and form matrices
```

```
      READ(5,10010) K, ND,NE, HKNOT
10010 FORMAT(I2,2I5,F10.3)
      WRITE(6,10020) K, ND,NE, HKNOT
10020 FORMAT(//'1  CURVS test  K =',I2,', ND = ',I5,', NE = ',I5,
     1 ', HKNOT = ',F8.4/)
C   Initialization
      CALL CURV1( K,CB,HKNOT, JG,G,  NY,LA,B, LOW,LOWF,JS, T )
C
      DO 10 I=1,ND
        JD = I
        READ(5,10030) Y(1),Y(2)
10030   FORMAT(2F10.6)
C   Add to matrices
        CALL CURV2( 1, K,CB,HKNOT, LOW,LOWF,T, Y,YO, JG,G,GLD,
     1    NY,LA,B,ALIB, LFLAG )
        IF(LFLAG.NE.0) GO TO 1001
        IF(I.LT.K) GO TO 10
C   Compute fit (open curve)
          CALL CURV3( 1, K,CB, LA, LOW,LOWF,JS, JG,G,GLD,
     1      NY,LA,B,ALIB,A, LCK,LCM,CPP, MK,  LFLAG )
          IF(LFLAG.NE.0) GO TO 1001
C   Print out fit
          DO 5 L=1,NY
    5       WRITE(6,10035) L, (A(J,L), J=1,JS)
10035       FORMAT(/' BASIS COEFFICIENTS...',I2/(6F12.5))
   10   CONTINUE
C   Compute final fit (closed curve)
      CALL CURV4( K,CB, LOW,JS, JG,G,  NY,LA,A,B,
     1  LCK,LCM,CPP, MK, LFLAG )
      IF(LFLAG.NE.0) GO TO 1001
C
C   Evaluate fit
C
      WRITE(6,10040)
10040 FORMAT(//'  Fit evaluation .. '/)
      C = (LOW+1)*HKNOT
      DO 20 I=1,NE
        TX = (I-1)*C/(NE-1)
        E1 = EVAL( K,MK,0.,HKNOT,LCK,CPP(1,1,1), TX )
        E2 = EVAL( K,MK,0.,HKNOT,LCK,CPP(1,1,2), TX )
   20   WRITE(6,10045) E1,E2
10045   FORMAT(2F12.5)
      STOP
C   ERROR
 1001 WRITE(6,10050)  LFLAG, JD, LOW, JS, T
10050 FORMAT(' ERROR   ',I10/' JD,LOW,JS ',3I5,', T = ',F8.2)
      STOP
      END
```

```
C  **************************************************************
C  *                       Module CURVS                         *
C  *            Fast Least Squares Spline Curve Fitting         *
C  *                        28 Aug 74                           *
C  **************************************************************
C  Entries:   CURV1, CURV2, CURV3, CURV4
C  Externals: from module PSQARS: SETUPS,ADDON,EXPAND,FOLD,CONVRT
C             from module ENSOLV: FACTOR,SOLVE
C
C  Reference:   S.C. Eisenstat, J.W. Lewis, M.H. Schultz,
C  "A Real-Time Algorithm for Least Squares Splines and
C  Its Application in Computer Aided Geometric Design",
C  research report #29, Department of Computer Science,
C  Yale University
C
C      The CURVS module is designed for incremental fitting of curves
C in two or more dimensions.  The subroutines minimize storage and
C execution time while enabling evaluation of the fit at any time
C during acquisition of data.  There are four routines:  CURV1 for
C initialization, CURV2 for data point processing, CURV3 for open
C curve fit computation, and CURV4 for closed curve fit computation.
C Both CURV3 and CURV4 return the resulting spline as a piecewise
C polynomial which may be evaluated efficiently using EVAL in the
C PSQARS module.
C
C  This module may be used in three basic modes:
C
C 1)  Sequential:  The fit is not required until all data are acquired.
C     Call CURV1 for initialization; call CURV2 once for each data
C     point; and call CURV3 or CURV4 to compute the fit.
C     G and GLD may use the same storage.
C     A, B, and ALIB may use the same storage.
C     The control variable JCON may be set to 0 or 1.  If JCON=1,
C     the factorization GLD is computed in CURV2 while the data
C     are being acquired; and if JCON=0, the entire factorization
C     is computed in CURV3.
C
C 2)  Incremental: The fit will be computed several times before the
C     complete set of data is acquired.
C     Call CURV1 for initialization and CURV2 once for each data point.
C     CURV3 may be called at any time to compute the fit.
C     G and GLD may use the same storage.
C     A, B  and ALIB may use the same storage.
C     If G and GLD use the same storage, set JCON=1,
C     otherwise set JCON=-1.
C     For an exact fit, (but requiring more computation time as the
C     curve grows), set JLAG = LA.   For an approximate, but nearly
C     exact fit, set JLAG = 2*K (approximately).
C
C 3)  Incremental closed: The fit will be computed several times
C     before the complete set of data is acquired; and after the
C     complete set of data is acquired, a closed curve fit
C     will be computed.   Call CURV1 for initialization and CURV2
C     once for each data point.   CURV3 may be called at any time
C     to compute the open curve fit.   To compute the closed
```

```
      RETURN
      END
C***CURV2******************************************************************
      SUBROUTINE CURV2(JCON, K,CB,HKNOT, LOW,LOWF,T, Y,YO, JG,G,GLD,
    1   NDIM,LA,B,ALIB, LFLAG )
      DIMENSION CB(K,K), JG(LA),G(1),GLD(1)
      DIMENSION B(LA,NDIM),ALIB(LA,NDIM)
      DIMENSION Y(NDIM), YO(NDIM)
      LFLAG = 0
C   Arc length
      Q = 0.
      DO 10 I=1,NDIM
        IF(T.LT.0.) GO TO 10
        Q = Q + (Y(I)-YO(I))**2
   10   YO(I) = Y(I)
      T = T + SQRT(Q)
      IF(T.LT.0) T = 0.
C   Compute interval
      JO = 1
      IF(JCON.LT.0) JO = K+1
   20 DX = T-LOW*HKNOT
      IF(DX.LT.HKNOT) GO TO 30
        LOW = LOW+1
        IF( (LOW+K+JO-1).LE.LA ) GO TO 20
          LFLAG = 1
          RETURN
C   Add to matrix
   30 JOG = JG(JO)+JO
      CALL ADDON(K,CB,DX,1.,NDIM,1,Y, LOW, JG,G(JOG),LA,B(JO,1))
C   Factor if necessary and requested
      IF(JCON.EQ.0) RETURN
      IF(LOWF.GE.LOW) RETURN
      CALL FACTOR( LOWF+1,LOW, JG,G(JOG),GLD,  NDIM,LA,B(JO,1),ALIB,
    1   LFLAG )
      LOWF = LOW
      RETURN
      END
C***CURV3******************************************************************
      SUBROUTINE CURV3( JCON, K,CB, JLAG, LOW,LOWF,JS, JG,G,GLD,
    1   NDIM,LA,B,ALIB,A, LCK,LCM,CPP, MK, LFLAG )
      DIMENSION CB(K,K), JG(LA),G(1),GLD(1)
      DIMENSION B(LA,NDIM),ALIB(LA,NDIM), A(LA,NDIM)
      DIMENSION CPP(LCK,LCM,NDIM)
      JS = LOW+K
C  Factor remainder of matrix
      JO = 1
      IF(JCON.LT.0) JO = K+1
      JOG = JG(JO)+JO
      LOWF = LOW
      IF(JCON.EQ.0) LOWF = 0
      IF((LOWF+1).LT.JS) CALL FACTOR(LOWF+1,JS-1, JG,G(JOG),GLD,
    1   NDIM,LA,B(JO,1),ALIB, LFLAG )
      IF(LFLAG.NE.0) RETURN
      CALL FACTOR( JS,JS,JG,G(JOG),GLD, NDIM,LA,B(JO,1),ALIB, MFLAG)
      IF(MFLAG.NE.0) JS = JS - 1
      MK = JS - K + 2
C  Solve
      JLOW = 1 + MAX0( 0, MIN0( JS, LOWF-JLAG ) )
      CALL SOLVE( JLOW,JS, JG,GLD, NDIM,LA,A,ALIB )
C  Convert
      CALL CONVRT( K,CB, JLOW,MK, NDIM,LA,A, LCK,LCM,CPP )
      RETURN
      END
C***CURV4******************************************************************
      SUBROUTINE CURV4( K,CB, LOW,JS, JG,G,  NDIM,LA,A,B,
    1   LCK,LCM,CPP, MK, LFLAG )
      DIMENSION CB(1), JG(LA), G(1), A(LA,NDIM), B(LA,NDIM)
      DIMENSION CPP(LCK,LCM,NDIM)
      MK = LOW+2
      JS = LOW+K
      LFLAG = 2
      IF(MK.LE.K) RETURN
C   Expand and fold matrix G
      CALL EXPAND( K, MK, JG,G )
      CALL FOLD( K, MK, JG,G, NDIM,LA,B )
C   Factor G
      JS = LOW+1
      CALL FACTOR( 1,JS, JG,G,G, NDIM,LA,B,A, LFLAG )
      IF(LFLAG.NE.0) RETURN
C   Solve
      CALL SOLVE(  1,JS, JG,G,   NDIM,LA,A,A )
      IF(K.LE.1) GO TO 20
C  Extend basis coefficients
        DO 10 L=1,NDIM
          DO 10 I=2,K
   10       A(I+MK-2,L) = A(I-1,L)
C  Convert
   20 CALL CONVRT( K,CB, 1,MK, NDIM,LA,A, LCK,LCM,CPP )
      RETURN
      END
C****END****CURVS**********************************************************
```

```
C      curve fit, call CURV4 after acquisition of the complete
C      set of data.    G, GLD, A, ALIB, and B must use different
C      storage.    Observe that the array G used for a closed curve is
C      DIMENSIONed nearly twice as large as that for an open curve.
C      JCON should be set to -1 and JLAG is chosen as in (2).
C
C      To achieve maximum efficiency, the user may wish to modify
C the organization of the modules CURVS, PSQARS, and IPSOLV.
C The current organization is designed for clarity rather than
C for efficiency.    For example, the subroutine call to CURV2
C may require more time than the computation in CURV2 itself.
C By incorporating some of the arguments in common blocks and
C eliminating some subroutine calls entirely, the user may speed
C up these routines considerably.
C     The same considerations apply to evaluation of the fit.
C The user interested in efficiency should not call the function
C EVAL to evaluate a spline; instead that calculation should be
C performed with in-line code.
C
C****************Subroutines and Calling Sequences*********************
C
C 1) Unless otherwise indicated, variable types are
C        IMPLICIT REAL(A-H, O-Z)
C        IMPLICIT INTEGER(I-N)
C 2) Value variables (V) pass a value to the subroutine and must
C    have been set by the user or by a previously called subroutine.
C    Result variables (R) return results to the calling subprogram.
C
C**CURV1: Setup subroutine for CURVS module
C       CALL CURV1( K,CB,HKNOT, JG,G,NDIM,LA,B, LOW,LOWF,JS,T )
C
C V  K           - Order of spline
C R  CB(K**2)    - The piecewise polynomial representation
C                  for the order k B-spline basis on uniform
C                  knots with spacing HKNOT
C V  HKNOT       - Knot spacing: if T is the total arc length,
C                  then LA must be GE T/HKNOT
C V  LA          - Dimension of JG,G,B
C R  JG(LA)      - Index array for grammian
C R  G(K,LA)     - Grammian (the profile of the lower triangle is stored)
C    G(2*K-1,LA) The first set of dimensions is sufficient for
C                  routines CURV1,CURV2,CURV3; but if CURV4 is to be
C                  called, then G must have the second dimensions
C                  NOTE: the indicated dimensions are for the
C                  purpose of allocating storage only
C                  The array is actually addressed as follows:
C                  G(I,J) = G( JG(I) + J, 1 )   for    I le J
C V  NDIM        - Dimension of curve
C R  B(LA,NDIM)- Right hand sides (NDIM of them, LA long)
C R  LOW         - Interval variable, (LOW*HKNOT) LE T LT ((LOW+1)*HKNOT)
C R  LOWF        - Value of LOW at last factorization of A
C R  JS          - Index of highest valid basis coefficient
C                  LOW,LOWF,JS are initialized to 0
C R  T           - Accumulated arc length (initialized to -1.)
C
```

```
C**CURV2
C  1) Computes the arc length T to data point Y(NDIM)
C  2) Adds contribution of data point Y(NDIM) to G and B
C  3) Factors G and performs forward solve on B, both up to row LOW
C
C        CALL CURV2( JCON, K,CB,HKNOT, LOW,LOWF,T, Y,YO,
C     1   JG,G,GLD, NDIM,LA,B,ALIB, LFLAG )
C
C V  JCON       - Control flag (set mode of operation)
C                  0 - Do not factor matrix G   (mode 1)
C                 -1 - Factor G; and CURV3 will be called (mode 2)
C                  1 - Factor G                   (mode 3)
C V  K,CB,HKNOT - As previously defined
C V  JG,NDIM,LA
C V  G,B
C VR LOW,LOWF,T
C
C V  Y(NDIM)     - Coordinates of point on curve
C VR YO(NDIM)    - Previous coordinates
C R  GLD(K,LA)- Factorization of G (if requested) to row LOW
C R  ALIB(LA)    - Forward solve of B to element LOW
C R  LFLAG       - Error code  (0, OK; 1, LA too small; -1, G  singular)
C
C**CURV3: Computes fit
C       CALL CURV3( JCON, K,CB, JLAG, LOW,LOWF,JS, JG,G,GLD,
C     1    NDIM,LA,B,ALIB,A, LCK,LCM,CPP, MK, LFLAG )
C
C V  JCON,K,CB,HKNOT - As previously defined
C V  JG,NDIM,LA
C V  LOW,G,B
C VR GLD,ALIB,LOWF,JS,
C R  LFLAG
C
C V  JLAG        - Number of intervals lower than LOWF to be backsolved
C                   (for  exact solution JLAG = LA)
C R  A(LA,K)     - Basis coefficients for resulting spline
C R  CPP(LCK,LCM,NDIM) - Piecewise polynomial for resulting spline
C R  MK          - Number of knots used
C
C**CURV4: Computes closed curve fit
C       CALL CURV4( K,CB, LOW, JG,G, NDIM,LA,B,A, LCK,LCM,CPP,MK, LFLAG)
C
C V  K,CB        - As previously defined
C V  NDIM,LA,LOW
C VR JG,G,B
C R  A,CPP,MK,LFLAG
C
C***CURV1****************************************************************
      SUBROUTINE CURV1( K,CB,HKNOT, JG,G,NDIM,LA,B, LOW,LOWF,JS, T )
      DIMENSION CB(K,K), JG(LA), G(1), B(LA,NDIM)
      CALL SETUPS( 0,K,CB,HKNOT, LA,  JG,G, NDIM,LA,B )
      LOW = 0
      LOWF = 0
      JS = 0
      T = -1.
```

```
C     ****************************************************************    PSQR
C     *                       Module PSQARS                          *    PSQR
C     *              Least Squares Polynomial Splines                *    PSQR
C     *                        Primitives                            *    PSQR
C     *                        22 Sep 74                             *    PSQR
C     ****************************************************************    PSQR
C     Entries:   SETUPS, ADDON, EXPAND, FOLD, CONVRT, EVAL                 PSQR
C     Externals: none beyond FORTRAN IV library                            PSQR
C     These subroutines are to be used in the incremental formation        PSQR
C     of the spline normal equations                                       PSQR
C                                                                          PSQR
C     Reference:   S.C. Eisenstat, J.W. Lewis, M.H. Schultz,               PSQR
C     "A Real-Time Algorithm for Least Squares Splines and                 PSQR
C     Its Application in Computer Aided Geometric Design",                 PSQR
C     research report #29, Department of Computer Science,                 PSQR
C     Yale University                                                      PSQR
C                                                                          PSQR
C     *************Subroutines and Calling Sequences*****************      PSQR
C                                                                          PSQR
C 1) Unless otherwise indicated, variable types are                        PSQR
C        IMPLICIT REAL(A-H, O-Z)                                           PSQR
C        IMPLICIT INTEGER(I-N)                                             PSQR
C 2) Value variables (V) pass a value to the subroutine                    PSQR
C    Result variables (R) return results to the calling subprogram         PSQR
C                                                                          PSQR
C**SETUPS                                                                  PSQR
C  This subroutine must be called before using ADDON or CONVRT             PSQR
C  1) Sets up basis function array CB                                      PSQR
C  2) Sets up index array JG                                               PSQR
C  3) Zeros parts of arrays G and B                                        PSQR
C                                                                          PSQR
C        CALL SETUPS( JPER,  K,CB,HMESH, NSET,  JG,G, NY,LB,B )            PSQR
C                                                                          PSQR
C V  JPER      - Periodic flag, if non-zero setup for periodic splines     PSQR
C V  K         - Order of spline                                           PSQR
C R  CB(K**2)  - The piecewise polynomial representation                   PSQR
C                for the order k B-spline basis on uniform                 PSQR
C                knots with spacing HMESH                                  PSQR
C V  HMESH     - Knot spacing                                              PSQR
C V  NSET      - Number of entries to be set in JG, G, and B               PSQR
C V  LB,NY     - Dimensions of B                                           PSQR
C R  G(*)      - Gram matrix (profile only is stored)                      PSQR
C                G(I,J) = G( JG(I) + J )   for   I le J                    PSQR
C R  JG(LB)    - Index array for Gram matrix                               PSQR
C R  B(LB,NY)  - Right hand sides (NY of them, LB long)                    PSQR
C                                                                          PSQR
C RESTRICTIONS:  K must be LE 6, NSET must be LE LB,                       PSQR
C                and dim(G) must be GE (NSET*K - K*(K-1)/2)                PSQR
C                                                                          PSQR
C**ADDON                                                                   PSQR
C  Adds contribution of measurements at points ( DX+LOW*HMESH, Y )         PSQR
C  to the matrices G and B                                                 PSQR
C                                                                          PSQR
C        CALL ADDON( K,CB, DX,W, NY,LY,Y, LOW, JG,G, LB,B )                PSQR
C                                                                          PSQR
C V  LOW, DX    - Data point is at  DX + LOW*HMESH + XLEFT                 PSQR
C                 where XLEFT is the left hand end point of the interval   PSQR
C V  W          - Data weight                                              PSQR
C V  NY,LY      - Dimensions of Y                                          PSQR
C V  Y(LY,NY)   - Data values (NY of them spaced at intervals LY in Y)     PSQR
C                                                                          PSQR
C V  JG(LB),LB,CB(K**2)  - As previously defined                           PSQR
C VR G(*),B(LB)                                                            PSQR
C                                                                          PSQR
C RESTRICTIONS: SETUPS must have been called first with NSET GE (LOW+K),   PSQR
C               and LB must be GE (LOW+K)                                  PSQR
C                                                                          PSQR
C**EXPAND                                                                  PSQR
C  Expands matrix G stored as a K wide band into the shape needed          PSQR
C  for the periodic matrix                                                 PSQR
C                                                                          PSQR
C        CALL FOLD( K, MK, JG,G )                                          PSQR
C                                                                          PSQR
C V  K,JG(MK+K-2)  - As previously defined                                 PSQR
C VR G(*)                                                                  PSQR
C                                                                          PSQR
C RESTRICTIONS: SETUPS must have been called first with NSET GE (MK+K-2)   PSQR
C               and dim(G) must be GE (2*K-1)*(MK+K-2)                     PSQR
C                                                                          PSQR
C**FOLD                                                                    PSQR
C  Folds non-periodic matrices to create periodic matrices.  EXPAND        PSQR
C  or the equivalent routine must have been called first                   PSQR
C                                                                          PSQR
C        CALL FOLD( K, MK, JG,G, NY,LB,B )                                 PSQR
C                                                                          PSQR
C V  K,JG(LB),NY,LB  - As previously defined                               PSQR
C VR G(*),B(LB)                                                            PSQR
C                                                                          PSQR
C RESTRICTIONS: (MK+K-2) must be LE LB,                                    PSQR
C               and dim(G) must be LE (MK+K-2)*(2*K-1)                     PSQR
C                                                                          PSQR
C**CONVRT                                                                  PSQR
C  Converts basis coefficients A into piecewise polynomial                 PSQR
C  coefficients CPP                                                        PSQR
C                                                                          PSQR
C        CALL CONVRT( K,CB, JLOW,JHIGH, NY,LA,A, LCK,LCM,CPP )             PSQR
C                                                                          PSQR
C V  JLOW,JHIGH        - Intervals for which PP is to be computed          PSQR
C V  LA,NY             - Dimensions of A                                   PSQR
C V  A(LA,NY)          - B-spline basis coefficients                       PSQR
C V  LCK,LCM,NY        - Dimensions of CPP                                 PSQR
C R  CPP(LCK,LCM,NY)- Array of piecewise polynomial coefficients           PSQR
C                                                                          PSQR
C K,CB                 - As previously defined                             PSQR
C                                                                          PSQR
C RESTRICTIONS: SETUPS must have been called first, K must be LE LCK,      PSQR
C               (JHIGH-1) must be LE LCM                                   PSQR
C                                                                          PSQR
C**EVAL                                                                    PSQR
C  Evaluates spline at XX from PP representation                           PSQR
```

```
C
C       VALUE = EVAL( K,MK,XLEFT,HMESH, LCK,CPP, XX )
C
C V  XX                   - Evaluation point
C V  K,MK,XLEFT,HMESH,LCK,CPP - As previously defined
C
C***SETUPS**************************************************************
      SUBROUTINE SETUPS( JPER, K,CB,HMESH, NSET,  JG,G, NY,LB,B )
      DIMENSION CB(K,K), JG(LB), G(1), B(LB,NY)
C
C  CBM is an array containing columns of the
C  K coefficients for each of the K different piecewise polynomials
C  of the K order B-spline basis functions on an integer mesh.
C  The entries for order K begin at CBM( JCBM(K)+1 ).
C  K must be less than KMAX = 6.
C
      DIMENSION JCBM(6), CBM(91), CBM1(30), CBM2(25), CBM3(36)
      EQUIVALENCE (CBM,CBM1), (CBM(31),CBM2), (CBM(56),CBM3)
      DATA JCBM/ 0,1,5,14, 30,55/
      DATA CBM1/
     1  1.,       1.,-1.,  0.,+1.,
     2  5.0000000E-01,-1.0000000E+00, 5.0000000E-01,
     3  5.0000000E-01, 1.0000000E+00,-1.0000000E+00,
     4  0.0000000E-01, 0.0000000E-01, 5.0000000E-01,
     5  1.6666667E-01,-5.0000000E-01, 5.0000000E-01,-1.6666667E-01,
     6  6.6666666E-01, 0.0000000E-01,-1.0000000E+00, 5.0000000E-01,
     7  1.6666667E-01, 5.0000000E-01, 5.0000000E-01,-5.0000000E-01,
     8  0.0000000E-01, 0.0000000E-01, 0.0000000E-01, 1.6666667E-01/
      DATA CBM2/
     1  4.1666667E-02,-1.6666667E-01, 2.5000000E-01,-1.6666667E-01,
     2  4.1666667E-02, 4.5833333E-01,-5.0000000E-01,-2.5000000E-01,
     3  5.0000000E-01,-1.6666667E-01, 4.5833333E-01, 5.0000000E-01,
     4 -2.5000000E-01,-5.0000000E-01, 2.5000000E-01, 4.1666667E-02,
     5  1.6666667E-01, 2.5000000E-01, 1.6666667E-01,-1.6666667E-01,
     6  0.0000000E-01, 0.0000000E-01, 0.0000000E-01, 0.0000000E-01,
     7  4.1666667E-02/
      DATA CBM3/
     1  8.3333333E-03,-4.1666667E-02, 8.3333333E-02,-8.3333333E-02,
     2  4.1666667E-02,-8.3333333E-03, 2.1666667E-01,-4.1666666E-01,
     3  1.6666667E-01, 1.6666667E-01,-1.6666667E-01, 4.1666667E-02,
     4  5.5000000E-01, 0.0000000E-01,-5.0000000E-01, 0.0000000E-01,
     5  2.5000000E-01,-8.3333333E-02, 2.1666667E-01, 4.1666666E-01,
     6  1.6666667E-01,-1.6666667E-01,-1.6666667E-01, 8.3333333E-02,
     7  8.3333333E-03, 4.1666667E-02, 8.3333333E-02, 8.3333333E-02,
     8  4.1666667E-02,-4.1666667E-02, 0.0000000E-01, 0.0000000E-01,
     9  0.0000000E-01, 0.0000000E-01, 0.0000000E-01, 8.3333333E-03/
C
C  Set up index array  JG
C
      KM1 = K-1
      JG(1) = 0
      DO 10 I=2,NSET
 10     JG(I) = JG(I-1) + MIN0(I,K) - 1
      IF(JPER.EQ.0) GO TO 40
      IF(KM1.LE.0) GO TO 40
      M = 0
      JF = NSET-2*K+2
      DO 20 I=1,KM1
        M = M + MAX0( 0, I+JF-K )
 20     JG(I+JF) = JG(I+JF) + M
      JF = JF+KM1
      DO 30 I=1,KM1
 30     JG(I+JF) = JG(I+JF) + M
C  Zero array G, and NY columns of B
 40   DO 50 L=1,NY
        DO 50 I=1,NSET
 50       B(I,L) = 0.
      NN = JG(NSET) + NSET
      DO 60 I=1,NN
 60     G(I) = 0.
C
C  Set up TABLES common
C
      F = 1.
      DO 80 I=1,K
        IJ = JCBM(K) +I
        DO 70 J=1,K
          CB(I,J) = CBM(IJ)*F
 70       IJ = IJ+K
 80     F = F/HMESH
      RETURN
      END
C***ADDON***************************************************************
      SUBROUTINE ADDON( K,CB, DX,W, NY,LY,Y, LOW, JG,G, LB,B )
      DIMENSION CB(K,K), Y(LY,NY), JG(LB), G(1), B(LB,NY)
      DIMENSION VB(10)
C  Evaluate basis functions; put K non-zero values in VB
      DO 20 I=1,K
        VV = 0.
        DO 10 J=1,K
 10       VV = DX*VV + CB(K-J+1,I)
 20     VB(I) = VV
C
      DO 70 I=1,K
C  Add into matrix G
        VBW = VB(I)*W
        II = JG(I+LOW)+LOW
        DO 40 J=1,I
 40       G(II+J) = G(II+J) + VBW*VB(J)
C  Add into B
        DO 60 L=1,NY
 60       B(I+LOW,L) = B(I+LOW,L) + Y(1,L)*VBW
 70     CONTINUE
      RETURN
      END
C***EXPAND**************************************************************
      SUBROUTINE EXPAND( K, MK, JG,G )
      DIMENSION JG(1), G(1)
      KM1 = K-1
      IF(KM1.LE.0) RETURN
```

```
      JF1 = MK-1
      JF2 = MK-K
      JGO = JG(MK+K-2) + MK+K-2
C   Expand and Copy
      M = 0
      LM = K*KM1
      DO 10 I=1,KM1
        LM = LM + MIN0( I+JF2, K )
        M = M + MAX0( 0, I+JF2-K )
   10   JG(I+JF2) = JG(I+JF2) + M
      IF(M.LE.0) RETURN
      DO 20 I=1,KM1
   20   JG(I+JF1) = JG(I+JF1) + M
      JGN = JGO+M
      DO 30 I=1,LM
   30   G(JGN-I+1) = G(JGO-I+1)
      II = JGN-LM
      DO 70 I=1,KM1
        JGI = JG(I+JF2)
        LL = JF2+I-K
        IF(LL.LE.0) GO TO 50
          DO 40 J=1,LL
   40       G(JGI+J) = 0.
   50   LN = MIN0( K, JF2+I )
        JGU = JGI+JF2+I-LN
        DO 60 J=1,LN
   60     G(JGU+J) = G(II+J)
   70   II = II+LN
      RETURN
      END
C***FOLD*****************************************************************
      SUBROUTINE FOLD( K, MK, JG,G, NY,LB,B )
      DIMENSION JG(LB), G(1), B(LB,NY)
      KM1 = K-1
      IF(KM1.LE.0) RETURN
      JF1 = MK-1
      JF2 = MK-K
C   Fold
      DO 30 I=1,KM1
        JGD1 = JG(I)
        JGS1 = JG(I+JF1)+JF1
        JGD2 = JG(JF2+I)
        DO 10 L=1,NY
   10     B(I,L) = B(I,L) + B(I+JF1,L)
        DO  20 J=1,I
          JGS2 = JG(JF1+J) + JF2+I
          G(JGD2+J) = G(JGD2+J) + G(JGS2)
   20     G(JGD1+J) = G(JGD1+J) + G(JGS1+J)
   30   CONTINUE
      RETURN
      END
C***CONVRT***************************************************************
      SUBROUTINE CONVRT( K,CB, JLOW,JHIGH, NY,LA,A, LCK,LCM,CPP )
      DIMENSION CB(K,K), A(LA,NY), CPP(LCK,LCM,NY)
      MM = JHIGH-1
      DO 30 L=1,NY
        DO 30 M=JLOW,MM
          DO 20 J=1,K
            DD = 0.
            DO 10 I=1,K
   10         DD = DD + A(I+M-1,L)*CB(J,I)
   20       CPP(J,M,L) = DD
   30   CONTINUE
      RETURN
      END
C***EVAL*****************************************************************
      FUNCTION EVAL( K,MK,XLEFT,HMESH, LCK,CPP, XX )
      DIMENSION CPP(LCK,1)
      LOW = MAX0( 0, MIN0( MK-2, INT( (XX-XLEFT)/HMESH )))
      DX  = XX - (XLEFT + LOW*HMESH)
      EVAL = 0.
      DO 10 I=1,K
   10   EVAL = EVAL*DX + CPP(K-I+1,LOW+1)
      RETURN
      END
C****END****PSQARS*******************************************************
```

```
C     ****************************************************************
C     *                     Module ENSOLV                            *
C     *  Incremental Symmetric Envelope Linear Equation Solver      *
C     *                     29 Aug 74                                *
C     ****************************************************************
C     Entries:   FACTOR, SOLVE
C     Externals: none beyond FORTRAN IV library
C
C     Reference:   S.C. Eisenstat, J.W. Lewis, M.H. Schultz,
C     "A Real-Time Algorithm for Least Squares Splines and
C     Its Application in Computer Aided Geometric Design",
C     research report #29, Department of Computer Science,
C     Yale University
C
C        The module ENSOLV computes solutions to linear systems A*X = B
C where A is an N x N positive definite matrix of which half of the
C envelope is stored; B is an arbitrary N x NB matrix, and X is the
C N x NB solution matrix.
C The system is solved in three steps:
C a) Factoring the matrix A as L*D*(L transpose), here called ALD,
C    where L is lower triangular with unit diagonal and D is diagonal
C b) Solving the triangular system  L*ALIB = B for ALIB
C c) Solving the traingular system  D*(L transpose)*X = ALIB for X
C
C     The matrix A and its factorization ALD are stored in one
C dimensional arrays and accessed through the index array JA:
C     A( I,J ) = A( JA(I) + J ) for I le J
C Only the lower triangle of the envelope of A and ALD is stored since
C all other elements are zero.  For example, consider the lower
C triangle of matrix A, its envelope, and its index array JA:
C
C       A: X         envelope(A): E            JA: 0
C          XX                     EE               1
C          0XX                    0EE              2
C          000X                   000E             2
C          X000X                  EEEEE            6
C          00XX0X                 00EEEE           9
C
C For a given row of the lower triangle, the envelope is composed
C of those elements in the row of same or higher column index than
C the nonzero element of lowest column index.  To compute the index
C vector JA, use the formula:
C
C     JA(1) = 0
C     JA(I) = JA(I-1)-1 + (number of elements in the envelope for row I)
C
C To solve a simple linear system, A*X = B
C       CALL FACTOR( 1,N, JA,A,A, 1,0,B,B,   LFLAG )
C       CALL SOLVE ( 1,N, JA,A,    1,0,X,B        )
C LFLAG is a status code, zero indicates successful completion.
C (in this example the factorization LD replaces the contents of A
C and (L inverse)*B replaces the contents of B )
C
C     ENSOLV may also provide incremental factorizations and solutions.
C Since the factorization of A is computed row by row,  no element of
C ALD in a given row depends on any element of A in subsequent rows;
C consequently the factorization for the first Q rows of a matrix
C may be computed without knowledge of the values for rows Q+1, Q+2,....
C For example:
C
C a)  COMPUTE K rows of the factorization LD (which replace the
C     corresponding rows of A) and K elements of (L inverse)*B
C     (which replace the corresponding elements of B)
C           CALL FACTOR( 1,K, JA,A,A, 1,0,B,B,   LFLAG )
C b)  COMPUTE K additional rows
C           CALL FACTOR( K+1,2*K, JA,A,A, 1,0,B,B,   LFLAG )
C c)  COMPUTE K additional rows
C           CALL FACTOR( 2*K+1,3*K, JA,A,A, 1,0,B,B,   LFLAG )
C
C The array A will contain its factorization LD in the first 3*K
C rows and subsequent rows will be unchanged.
C
C***************Subroutines and Calling Sequences*********************
C
C 1) Unless otherwise indicated, variable types are
C        IMPLICIT REAL(A-H, O-Z)
C        IMPLICIT INTEGER(I-N)
C 2) Value variables (V) pass a value to the subroutine
C    Result variables (R) return results to the calling subprogram
C
C**FACTOR
C  1) Factors A = L*D*(L transpose)
C  2) Forward solves ALIB = (L inverse)*B
C
C CALL:
C        CALL FACTOR( JBOT,JTOP, JA,A,ALD, NB,LB,B,ALIB, LFLAG )
C
C V  JBOT, JTOP- Range over which envelope solution is carried out
C V  A(*)       - The envelope of the matrix A is stored in row order
C V  JA(LB)     - Index array for A.  To get an element:
C                   A( I,J ) = A( JA(I) + J )  for I le J.
C                 The number of elements of row I  belonging to the
C                 envelope is     JA(I) - JA(I-1) + 1.
C                 Length of matrix = JA(N) + N for NxN matrix.
C V  B(LB,NB)   - Right hand sides (NB of them, LB long)
C
C R  ALD(*)   - L D LT decomposition of A
C R  ALIB(LB,NB)-L inverse times B
C
C R  LFLAG is -1 if A is NOT positive definite, 0  otherwise
C
C  A and ALD may use the same storage
C  B and ALIB  may use the same storage
C
C**SOLVE
C  Back solves  X = (LT inverse)*(D inverse)* ALIB
C
C        CALL SOLVE( JBOT,JTOP, JA,ALD, NB,LB,X,ALIB )
C
C V  JBOT,JTOP,NB,LB  - As previously defined
```

```
C    ALD(*), ALIB(*)
C R  X(LB,NB)  - Solution vectors (NB of them, LB long)
C
C  X and ALIB may use the same storage
C
C***FACTOR*************************************************************
      SUBROUTINE FACTOR( JBOT,JTOP,  JA,A,ALD, NB,LB,B,ALIB, LFLAG )
      DIMENSION  JA(1), A(1), ALD(1), B(1), ALIB(1)
C  EPS is the single precision rounding error (here for the PDP10)
      DATA EPS/1.E-7/
      LFLAG = 0
C   Factorization
      DO 110 K=JBOT,JTOP
        JAK = JA(K)
        IMAX = K-1
        IF(IMAX.LE.0)   GO TO 50
        IMIN = MAX0( 1, K-JA(K)+JA(K-1) )
        IF (IMIN.GE.K)  GO TO 50
C   Off diagonal elements
        DO 40 I=IMIN,IMAX
          JMAX  = I-1
          AIK = A(JAK+I)
          IF(JMAX.LE.0) GO TO 40
          JAI = JA(I)
          JMIN = MAX0 (I-JA(I)+JA(I-1), IMIN)
          IF (JMIN.GE.I)  GO TO 40
          DO 30 J=JMIN,JMAX
   30       AIK = AIK - ALD(JAI+J) * ALD(J+JAK)
   40     ALD(I+JAK) = AIK
C   Diagonal elements
   50   ALKK = A(K+JAK)
        IF (IMAX.LE.0)  GO TO 70
        IF (IMIN.GE.K)  GO TO 70
        DO 60 I=IMIN,IMAX
          JAI = JA(I)
          AIK = ALD(I+JAK)
          ALIK = AIK*ALD(I+JAI)
          ALD(I+JAK) = ALIK
   60     ALKK = ALKK - AIK*ALIK
C   Check sign of diagonal element (including rounding error)
   70   IF(ALKK.LE.AMAX1(0.,(K-1)*EPS*A(K+JAK))) GO TO 1001
        ALD(K+JAK) = 1./ALKK
C   Forward solve
        II = 0
        DO 100 J=1,NB
          BK = B(K+II)
          IF (IMAX.LE.0)  GO TO 90
          IF (IMIN.GE.K)  GO TO 90
            DO 80 I=IMIN,IMAX
   80         BK = BK - ALD(JAK+I)*ALIB(I+II)
   90       ALIB(K+II) = BK
  100     II = II+LB
  110     CONTINUE
      RETURN
C   Error return:  not positive definite ( Di < 0 )
 1001 LFLAG = -1
      RETURN
      END
C***SOLVE**************************************************************
      SUBROUTINE SOLVE( JBOT,JTOP, JA,ALD, NB,LB,X,ALIB )
      DIMENSION  JA(1), ALD(1), X(1), ALIB(1)
C
      II = 0
      DO 40 J=1,NB
C  Multiply by D inverse
        DO 10 K=JBOT,JTOP
          JAK = JA(K)
   10     X(K+II) = ALD(K+JAK) * ALIB(K+II)
        MM = JTOP-JBOT
        IF(MM.LE.0) GO TO 40
C   Back solve    X = (L transpose inverse)*(D inverse)*B
        K = JTOP
        DO 30 L=1,MM
          JAK = JA(K)
          IMIN = MAX0( 1, K-JA(K)+JA(K-1) )
          IF (IMIN.GE.K)  GO TO 30
          IMAX = K-1
          DO 20 I=IMIN,IMAX
   20       X(I+II) = X(I+II) - ALD(I+JAK)*X(K+II)
   30     K = K-1
   40   II = II+LB
      RETURN
      END
C****END****ENSOLV*****************************************************
```

Appendix II: PP-Representation of the B-Splines

For t satisfying

(II.1) $i \cdot h \le t < (i+1) \cdot h$

and with

(II.2) $\delta = t - ih$

the B-splines may be written as

(II.3) $$N_{i+1,k}(t) = \sum_{\ell=0}^{k-1} C^N_{1\ell} \, \delta\ell$$

$$\vdots$$

$$N_{i+k,k}(t) = \sum_{\ell=0}^{k-1} C^N_{k\ell} \, \delta\ell$$

$$N_{j,k}(t) = 0 \qquad \text{for } j \le i \text{ or } j > i+k.$$

A table of the coefficients C^N_{ij} follows. This table was generated by rationalizing the numbers generated by the program PPBAS. If higher order splines are desired, PPBAS may be used to extend this table.

Order	δ^0	δ^1	δ^2	δ^3	δ^4	δ^5
	Coefficients					
1	1					
2	1	-1				
	0	1				
3	1/2	-1	1/2			
	1/2	1	-1			
	0	0	1/2			
4	1/6	-1/2	1/2	-1/6		
	2/3	0	-1	1/2		
	1/6	1/2	1/2	-1/2		
	0	0	0	1/6		
5	1/24	-1/6	1/4	-1/6	1/24	
	11/24	-1/2	-1/4	1/2	-1/6	
	11/24	1/2	-1/4	-1/2	1/4	
	1/24	1/6	1/4	1/6	-1/6	
	0	0	0	0	1/24	
6	1/120	-1/24	1/12	-1/12	1/24	-1/120
	13/60	-5/12	1/6	1/6	-1/6	1/24
	11/20	0	-1/2	0	1/4	-1/12
	13/60	1/24	1/6	-1/6	-1/6	1/12
	1/120	1/24	1/12	1/12	1/24	-1/24
	0	0	0	0	0	1/120

B-Spline Piecewise Polynomials

```
C     ****************************************************************
C     *                      Program PPBAS                            *
C     *               B-spline Piecewise Polynomials                  *
C     *                       29 Aug 1974                             *
C     ****************************************************************
C     EXTERNALS: None
C
C     Reference:   S.C. Eisenstat, J.W. Lewis, M.H. Schultz,
C     "A Real-Time Algorithm for Least Squares Splines and
C     Its Application in Computer Aided Geometric Design",
C     research report #29, Department of Computer Science,
C     Yale University
C
C        PPBAS is a program to compute the piecewise polynomial
C    representations for the B-spline basis over a uniform mesh.
C
C     Do not change KMAX without changing array dimensions
         DATA KMAX/10/
         DATA LT/20/, NY/1/, LA/10/, LCK/10/, LCM/1/
         DIMENSION A(10), T(20), PP(10), XP(2)
C     Set knots
         KK = 2*KMAX
         DO 10 I=1,KK
   10      T(I) = FLOAT(i)
C     Set basis coefficients
         DO 20 I=1,KMAX
   20      A(I) = 0.
C
C     Compute and Print piecewise polynomial
C
         WRITE(6,10010) KMAX, (I, I=1,KMAX)
10010    FORMAT('  Piecewise Polynomials for B-splines to Order ',
    1    I3/'  Order             Coefficients'/(7X,6I11))
C     Loop through K
         DO 130 K=1,KMAX
           WRITE(6,11010) K
11010      FORMAT(I5)
C     Loop through non-vanishing basis functions
           DO 120 L=1,K
             A(L) = 1.
             CALL CONVRT( K, LT,T, NY,LA,A,K, LCK,LCM,XP,PP, NXP,
    1          LFLAG )
             IF( LFLAG.NE.0 ) GO TO 200
C     Convert derivatives at knots to polynomial coefficients
             F = 1.
             DO 110 I=1,K
               PP(I) = PP(I)*F
  110          F = F/FLOAT(I)
C     Print results
             WRITE(6,11020)  (PP(J), J=1,K)
  120        A(L) = 0.
  130      CONTINUE
11020      FORMAT(10X,6F11.7)
         STOP
C     Error in CONVRT
  200    WRITE(6,12010) LFLAG
12010    FORMAT(' ? CONVRT ERROR # 'I5)
         STOP
         END
C***CONVRT***************************************************************
C Computes piecewise polynomial from basis coefficients
C
C        CALL CONVRT( K, LT,T, NY,LA,A,NA, LCK,LCM,XP,DPP, NXP, LFLAG )
C
C    Reference:  J.W. Lewis, M.H. Schultz, S.C. Eisenstat
C    "FITS: A Subroutine Package for Spline Regression",
C    research report, Department of Computer Science, Yale University
C
C 1) Unless otherwise indicated, variable types are
C        IMPLICIT REAL(A-H, O-Z)
C        IMPLICIT INTEGER(I-N)
C 2) Value variables (V) pass a value to the subroutine and must
C    have been set by the user or by a previously called subroutine.
C    Result variables (R) return results to the calling subprogram.
C
C V  K              - Degree +1 of spline  (less than 10)
C V  T(LT)          - B-spline knots
C V  A(LA,NY)       - B-spline basis coefficients
C V  NA             - Number of B-spline coefficients
C R  XP(LCM+1)      - Knots
C R  DPP(LCK,LCM,NY)- Piecewise polynomial stored as the K derivatives
C                     of the spline at each knot XP(I) for  I=1,NXP
C R  NXP            - Number of intervals+1
C R  LFLAG          - Status
C                      0 - OK
C                      1 - K GT KMAX  (KMAX = 10)
C                      2 - K GT LCK
C                      3 - NXP GT LCM
C
         SUBROUTINE CONVRT( K, LT,T, NY,LA,A,NA, LCK,LCM,XP,DPP, NXP,
    1      LFLAG )
         DIMENSION T(LT), A(LA,NY), XP(LCM), DPP(LCK,LCM,NY)
C    The dimensions of the following arrays determine KMAX
         DIMENSION AT(10,10), VB(10), DP(10), DM(10)
         DATA KMAX/10/
         KM1 = K-1
         LFLAG = 0
         IF( K.GT.LCK )  GO TO 1001
         IF( K.GT.KMAX )  GO TO 1002
         DO 400 JNY=1,NY
           JR = 0
           JTO = 0
           NXP = 0
           DO 300 JT=K,NA
             IF( T(JT).GE.T(JT+1) )  GO TO 300
               NXP = NXP+1
               IF( NXP.GT.LCM )  GO TO 1003
               XP(NXP) = T(JT)
C    Difference basis coefficients
               JD = JT-JTO
```

```
      DO 20 J=1,JD
        JP = JR+1
        IF(JP.GT.K) JP = 1
        JTOJ = JTO + J
        AT(1,JP) = A(JTOJ,JNY)
        LU = K-JD+J-1
        IF(LU.LE.0) GO TO 20
          DO 10 LL=1,LU
            KLL = K-LL
10          AT(LL+1,JP) = ( AT(LL,JP) - AT(LL,JR) )
 1            /( T(JTOJ+KLL) - T(JTOJ) ) * KLL
20      JR = JP
C   Find derivatives at knot
30    L = K
      VB(1) = 1.
      DO 220 LK=1,K
C   Evaluate basis functions of order LK at T(JT)
100     LKM1 = LK-1
        IF( LKM1.LE.0 ) GO TO 200
          DP(LKM1) = T(JT+LKM1) - T(JT)
          DM(LKM1) = T(JT) - T(JT-LKM1+1)
          VMP = 0.
          DO 110 I=1,LKM1
            VM = VB(I)/(DP(I) + DM(LK-I))
            VB(I) = VM*DP(I) + VMP
110         VMP = VM * DM(LK-I)
120       VB(LK) = VMP
C   Compute (L-1) derivative at XP(JT) from AT(L,*) and VB(*)
200   V = 0.
      JP = JR
      DO 210 J=L,K
        V = V + AT(L,JP)*VB(LK-J+L)
        JP = JP-1
        IF(JP.LE.0) JP = K
210     CONTINUE
      DPP(L,NXP,JNY) = V
220   L = L-1
      JTO = JT
300   CONTINUE
400   CONTINUE
      RETURN
C   Error returns
1003  LFLAG = LFLAG+1
1002  LFLAG = LFLAG+1
1001  LFLAG = LFLAG+1
      RETURN
      END
```