Pergamon

Comput. & Graphics, Vol. 18, No. 3, pp. 327–334, 1994

Printed in Great Britain
0097-8493/94 $6.00 + .00

0097-8493(94)E0015-P

Technical Section

A FAST ALGORITHM FOR CUBIC B-SPLINE CURVE FITTING

KUO-LIANG CHUNG
Department of Information Management, National Taiwan Institute of Technology, Taipei, Taiwan 10672, R.O.C.

and

WEN-MING YAN
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan 10764, R.O.C.

Abstract—Based on the matrix perturbation technique, a fast-fitting algorithm using uniform cubic B-spline curves is presented. Our algorithm entails much less floating-point operations when compared with Gaussian elimination method. In addition, our result can be applied to solve the closed cubic B-spline curve-fitting problem. Experimental results are included for a practical version. These experimental values confirm our theoretic results.

1. INTRODUCTION

Curve fitting is important in computer graphics, computer-aided design, pattern recognition, and picture processing[11, 13, 15]. The task of curve fitting is to construct a smooth curve that fits a set of given points in the space.

In practice, a curve-fitting algorithm should meet two criterions: First, adjusting a control point of the curve affects only its vicinity, and second, it should be fast enough to be incorporated into an interactive program. The cubic B-spline curve interpolation[11] is a good fitting tool to meet the first criterion. Gaussian elimination has been used to solve the cubic B-spline curve-fitting problem. How to speed up the computation of the cubic B-spline curve-fitting in order to meet the second criterion is a very interesting research problem. This paper only considers the uniform cubic B-spline case[2].

Based on the matrix perturbation technique, a fast-fitting algorithm using uniform cubic B-spline curves is presented. Given n points, the number of floating-point (FP for short) operations required for our algorithm is about $5n$. While using Gaussian elimination, it takes about $7n$ FP operations to solve the same fitting problem. Our algorithm entails much less FP operations when compared with Gaussian elimination method. In addition, our result can be applied to solve the closed cubic B-spline curve-fitting problem, and the number of FP operations used in our algorithm over the number of FP operations used in Gaussian elimination is about one-third. Experimental results are included for a practical version. These experimental values confirm our theoretic results derived in this paper.

2. THE CUBIC B-SPLINE CURVE FITTING

Suppose we are given a set of points, $B_i = (b_i^{(1)}, b_i^{(2)}, b_i^{(3)})$ for $1 \le i \le n$. According to [2], for an uniform cubic B-spline curve, each given point can be expressed by a weighted average of three control points:

$$B_i = \tfrac{1}{6}(C_{i-1} + 4C_i + C_{i+1}), \quad 1 \le i \le n,$$

where $C_i = (c_i^{(1)}, c_i^{(2)}, c_i^{(3)})$. They form a system of n equations in $n + 2$ unknowns for all given points. In order to completely solve the system, we need the following two additional equations to specify how the boundary control points are interpolated: $C_0 = C_1$; $C_{n+1} = C_n$. For simplicity, we only consider $\mathbf{b} = (b_1, b_2, \ldots, b_n)^t = (b_1^{(1)}, b_2^{(1)}, \ldots, b_n^{(1)})^t$ and $\mathbf{c} = (c_1, c_2, \ldots, c_n)^t = (c_1^{(1)}, c_2^{(1)}, \ldots, c_n^{(1)})^t$ throughout this paper. In what follows, matrices are represented by bold uppercase letters, vectors by bold lowercase letters, and scalars by plain lowercase letters. Thus, the above system of equations can be equivalently transformed into

$$\begin{pmatrix} 5 & 1 & 0 & & & \\ 1 & 4 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 \\ & & & 0 & 1 & 5 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{pmatrix} = 6 \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix} \leftrightarrow A\mathbf{c} = 6\mathbf{b}. \quad (1)$$

First, it takes n FP operations to perform the multiplication $6\mathbf{b}$. Using Gaussian elimination, the first is the forward-elimination phase, where Eq. (1) is transformed, by eliminating variables from equations, into a system with all zeros below the diagonal. It takes about $4n$ FP operations to perform this triangulation phase. At this moment, the coefficient matrix A becomes an unit upper-triangular matrix. The second

phase is the backward-substitution phase, where the values of the variables are computed using the triangulated matrix produced by the first phase. In this phase, it takes about $2n$ FP operations. Totally, solving Eq. (1) takes about $7n$ FP operations by using Gaussian elimination. The C language code of the cubic B-spline curve fitting algorithm using Gaussian elimination is given in Appendix A.

After solving the tridiagonal system of Eq. (1), we can obtain the control points, C_i, of the $n + 2$ defining polygon vertices. Letting $P_i(t)$ be the position vectors along the ith piecewise cubic curve as a function of the parameter t, the ith cubic B-spline curve segment is given by

$$P_i(t) = \sum_{j=-1}^{2} C_{i+j} N_j(t) \quad \text{for} \quad 1 \le i \le n-1 \qquad (2)$$

$$\text{and} \quad 0 \le t < 1,$$

where the $N_j(t)$ are the normalized B-spline blending functions. By the Cox-de Boor formulas[2], these periodic uniform basis functions in Eq. (2) are defined by: $N_{-1}(t) = \dfrac{-t^3 + 3t^2 - 3t + 1}{6}$; $N_0(t) = \dfrac{3t^3 - 6t^2 + 4}{6}$; $N_1(t) = \dfrac{-3t^3 + 3t^2 + 3t + 1}{6}$; $N_2(t) = \dfrac{t^3}{6}$.

Given 10 points denoted by the "star" symbols, by Eqs. (1) and (2), the corresponding control points denoted by the "circle" symbols and the curve interpolation denoted by a boldfaced line are illustrated in Fig. 1, where no curve end condition[1] is included to handle the two end segments of the curve.

3. A FAST ALGORITHM

Based on the matrix perturbation technique, this section presents a new three-phase algorithm for solving Eq. (1). It will be shown that our algorithm entails much less FP operations when compared with Gaussian elimination method. Due to the coefficient matrix A in Eq. (1) being near-Toeplitz, consider a perturbed matrix of A,

$$A' = \begin{pmatrix} a & 1 & & & & \\ 1 & 4 & 1 & & & \\ & \cdot & \cdot & \cdot & & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 4 \end{pmatrix} = L'U',$$

where

$$L = \begin{pmatrix} 1 & & & & & \\ -b & 1 & & & & \\ & \cdot & \cdot & & & \\ & & & -b & 1 & \\ & & & & -b & 1 \end{pmatrix} \quad \text{and}$$

$$U = \begin{pmatrix} a & 1 & & & & \\ & a & 1 & & & \\ & & \cdot & \cdot & & \\ & & & & a & 1 \\ & & & & & a \end{pmatrix},$$

then it implies that $a - b = 4$ and $-ab = 1$. By solving the two equations, we obtain $a = 2 + \sqrt{3}$ and $b = \sqrt{3} - 2$. It is clear that

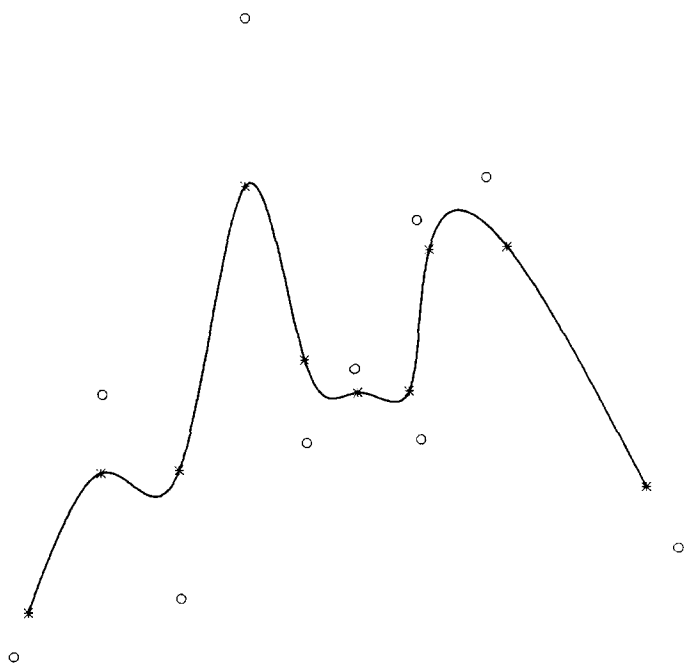

Fig. 1. An example of the cubic B-spline curve fitting.

$$A = A' + \begin{pmatrix} 1-b & & \\ & & \\ & & 1 \end{pmatrix}. \tag{3}$$

The above Toeplitz factorization procedure is called the first phase in ours algorithm, and it can be finished in $O(1)$ time. To solve $A\mathbf{c} = 6\mathbf{b}$, we first solve $A'\mathbf{c}' = 6\mathbf{b}$ $(= \mathbf{b}')$. It can be solved by the following forward and backward substitution procedure; it is also called the second phase in our algorithm, where $b = -\frac{1}{a}$ because $-ab = 1$.

```
for i=1 to n do b'_i = 6b_i
c'_1 = b'_1
for i=2 to n do c'_i = b'_i + b*c'_{i-1}
c'_n = -b*c'_n
for i=n-1 downto 1 do c'_i = b*(c'_{i+1} - b'_i)
```

It is not hard to verify that the number of FP operations required in the above procedure, *i.e.*, in the second phase, is about $5n$. By Eq. (3) and $\mathbf{b}' = 6\mathbf{b}$, we obtain

$$\begin{aligned} A\mathbf{c}' &= A'\mathbf{c}' + (1-b)c'_1\mathbf{e}_1 + c'_n\mathbf{e}_n \\ &= 6\mathbf{b} + (1-b)c'_1\mathbf{e}_1 + c'_n\mathbf{e}_n, \end{aligned} \tag{4}$$

where $\mathbf{e}_1 = \underbrace{(1, 0, \ldots, 0)^t}_{n}$ and $\mathbf{e}_n = \underbrace{(0, 0, \ldots, 0, 1)^t}_{n}$

Hence, it yields

$$\mathbf{c} = \mathbf{c}' - (1-b)c'_1A^{-1}\mathbf{e}_1 - c'_nA^{-1}\mathbf{e}_n.$$

It can be easily verified that $A\mathbf{c} = 6\mathbf{b}$.

Solving the recurrence relations: $x_{i-1} + 4x_i + x_{i+1} = 0$ for $2 \le i \le n-1$, we obtain $x_i = \beta b^i + \gamma\left(-\frac{1}{b}\right)^i$ for some constants β and γ. If $\gamma \neq 0$, then the value of x_i will become too large for sufficiently large i. In order to derive an approximated solution, temporarily, we try $x_i = \beta b^i$. By the first equation $5x_1 + x_2 = 1$, we have $x_i = \frac{b^i}{b-1}$ for $1 \le i \le n$. When i is sufficiently large, say, $i = n$, $x_i \approx 0$ and the last equation $x_{n-1} + 5x_n \approx 0$.

Since the sequence (x_i) converges to zero soon, let

$$x_i = \begin{cases} \dfrac{b^i}{b-1} & \text{if} \quad 1 \le i \le p (p\text{: a small integer}) \\ 0 & \text{if} \quad p+1 \le i \le n \end{cases}$$

and $\mathbf{x} = (x_1, x_2, \ldots, x_n)^t$, then we have

$$A\mathbf{x} = \mathbf{e}_1 - \frac{b^{p+1}}{b-1}\mathbf{e}_p + \frac{b^p}{b-1}\mathbf{e}_{p+1}. \tag{5}$$

Note that since $b = -0.2679492\ldots$, the last two terms of the right hand side of Eq. (5) can be negligible when $p \ge 10$. Hence $\mathbf{x}$ is a good approximation of $A^{-1}\mathbf{e}_1$ when $p \ge 10$. Similarly, let

$$y_i = \begin{cases} \dfrac{b^{n+1-i}}{b-1} & \text{if} \quad n-p+1 \le i \le n \\ 0 & \text{if} \quad 1 \le i \le n-p \end{cases}$$

and $\mathbf{y} = (y_1, y_2, \ldots, y_n)^t$, then we have

$$A\mathbf{y} = \mathbf{e}_n - \frac{b^{p+1}}{b-1}\mathbf{e}_{n+1-p} + \frac{b^p}{b-1}\mathbf{e}_{n-p}. \tag{6}$$

Let

$$\begin{aligned} \mathbf{c} &= \mathbf{c}' - c'_1(1-b)\mathbf{x} - c'_n\mathbf{y} \\ &= \mathbf{c}' + c'_1(\underbrace{b, b^2, \ldots, b^p}_{p}, \underbrace{0, \ldots, 0}_{n-p})^t \\ &\quad - \frac{c'_n}{b-1}(\underbrace{0, \ldots, 0}_{n-p}, \underbrace{b^p, \ldots, b^2, b}_{p})^t, \end{aligned}$$

which will be performed in the update phase (the third phase in our algorithm), thus is yields

$$\begin{aligned} A\mathbf{c} - 6\mathbf{b} = &-c'_1b^{p+1}\mathbf{e}_p + c'_1b^p\mathbf{e}_{p+1} \\ &+ \frac{c'_n}{b-1}b^{p+1}\mathbf{e}_{n+1-p} - \frac{c'_n}{b-1}b^p\mathbf{e}_{n-p}. \end{aligned} \tag{7}$$

Under a satisfactory residual requirement, say, $\|A\mathbf{c} - 6\mathbf{b}\|$ is of order 10^{-3}, how to determine the value of p to satisfy the residual requirement depends on Eq. (7) and the following Lemma.

Lemma 1. $\|\mathbf{c}'\| \le 3\|\mathbf{b}\|$, where $\|\mathbf{x}\| = \max_{1\le i\le n}(|x_i|)$.

Proof: Suppose $\|\mathbf{c}'\| > 3\|\mathbf{b}\|$. If $|c'_i| = \|\mathbf{c}'\|$, for some i $(2 \le i \le n-1)$, we have $c'_{i-1} + 4c'_i + c'_{i+1} = 6b_i$. Thus it gives $|4c'_i| \le |6b_i| + |c'_{i-1}| + |c'_{i+1}|$, *i.e.*,

$$\begin{aligned} |6b_i| &\ge |4c'_i| - |c'_{i-1}| - |c'_{i+1}| \\ &\ge 4\|\mathbf{c}'\| - \|\mathbf{c}'\| - \|\mathbf{c}'\| = 2\|\mathbf{c}'\| > 6\|\mathbf{b}\|. \end{aligned}$$

It is a contradiction.

If $|c'_i| = \|\mathbf{c}'\|$ for $i = 1$, we have $\left(-\frac{1}{b}\right)c'_1 = 6b_1 - c'_2$, and $\left|\left(-\frac{1}{b}\right)c'_1\right| \le |6b_1| + |c'_2|$. Thus it gives

$$\begin{aligned} |6b_1| &\ge \left|\left(-\frac{1}{b}\right)c'_1\right| - |c'_2| \ge \left(-\frac{1}{b}\right)\|\mathbf{c}'\| - \|\mathbf{c}'\| \\ &= (1+\sqrt{3})\|\mathbf{c}'\| > 3(1+\sqrt{3})\|\mathbf{b}\|. \end{aligned}$$

It is a contradiction.

If $|c'_i| = \|\mathbf{c}'\|$ for $i = n$, we have $4c'_n = 6b_n - c'_{n-1}$, and $|4c'_n| \le |6b_n| + |c'_{n-1}|$. Thus it gives

$$|6b_n| \geq |4c'_n| - |c'_{n-1}| \geq 4\|\mathbf{c}'\| - \|\mathbf{c}'\|$$
$$= 3\|\mathbf{c}'\| > 9\|\mathbf{b}\|.$$

It is a contradiction. We complete the proof. **Q. E. D.**

From Eq. (7) and Lemma 1, assuming that p, $p+1$, $n+1-p$, and $n-p$ are distinct, then four three-phase algorithm has the residual

$$\|A\mathbf{c} - 6\mathbf{b}\|$$
$$\leq \max\left(|c'_1 b^{p+1}|, |c'_1 b^p|, \left|\frac{c'_n}{b-1} b^{p+1}\right|, \left|\frac{c'_n}{b-1} b^p\right|\right)$$
$$= \max\left(|c'_1 b^p|, \left|\frac{c'_n}{b-1} b^p\right|\right)$$
$$\leq \max\left(3\|\mathbf{b}\|\,|b^p|, 3\|\mathbf{b}\|\left|\frac{b^p}{b-1}\right|\right) = 3\|\mathbf{b}\|\,|b^p|$$

That is, we have

$$\|A\mathbf{c} - 6\mathbf{b}\| \leq 3\|\mathbf{b}\|\,|b^p|.$$

For example, if $\|\mathbf{b}\| \leq 1000$ and $p = 10$, then the residual $\leq 5.7 \times 10^{-3}$. Under this conditions, in our implementation, the residual for the solution $\mathbf{c}(n > 10)$ is about of order 10^{-3}. Totally, it takes about $5n$ FP operations to solve Eq. (1) using our three-phase algorithm. Our algorithm entails much less FP operations when compared with Gaussian elimination method. Note that it takes $7n$ FP operations to solve Eq. (1) by using Gaussian elimination. The C language code of our fast cubic B-spline curve fitting algorithm is given in Appendix B.

Table 1 shows the performances of running our algorithm and the one using Gaussian elimination on IBM-386 personal computer (PC for short), where the symbol "s" denotes the time unit "second."

It is observed that our three-phase algorithm for solving Eq. (1) is faster than the method using Gaussian elimination. The value of "ratio" denotes the time spent on our algorithm over the time spent on Gaussian elimination method. The value of "ratio" is near to the theoretic value 5/7.

4. APPLICATION TO CLOSED CUBIC B-SPLINE CURVE FITTING

In this section, the application of the closed cubic B-spline curve fitting is investigated. Following the notations used in Section 2 and the definitions in [2], in the closed cubic B-spline curve fitting, each given point can be expressed by a weighted average of three control points:

$$B_1 = \tfrac{1}{6}(C_n + 4C_1 + C_2),$$
$$B_n = \tfrac{1}{6}(C_{n-1} + 4C_n + C_1), \quad \text{and}$$
$$B_i = \tfrac{1}{6}(C_{i-1} + 4C_i + C_{i+1}) \quad \text{for} \quad 2 \leq i \leq n-1.$$

They form a system of n equations in n unknowns for all given points. The above system of equations can be equivalently transformed into

$$\begin{pmatrix} 4 & 1 & & & & 1 \\ 1 & 4 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 \\ 1 & & & & 1 & 4 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{pmatrix}$$
$$= 6\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix} \leftrightarrow T\mathbf{c} = 6\mathbf{b}. \quad (8)$$

First, it takes n FP operations to perform the multiplication $6\mathbf{b}$. Using Gaussian elimination, the first is the triangulation phase, where Eq. (8) is transformed, by eliminating variables from equations, into a system with all zeros below the diagonal. It takes about $10n$ FP operations to perform this triangulation phase. The second phase is the backward-substitution phase, where the values of the variables are computed using the unit-triangulated matrix produced by the first phase. In this phase, it takes about $4n$ FP operations. Totally, it takes about $15n$ FP operations to solve Eq. (8) using Gaussian elimination.

After solving the circulant tridiagonal system of Eq. (8), we can obtain the control points c_i of the n defining polygon vertices. Given 10 given points denoted by the "star" symbols, by Eq. (8) and the similar definition in Eq. (2), the corresponding control points denoted by the "circle" symbols and the curve interpolation denoted by a boldfaced line are illustrated in Fig. 2.

Following our three-phase approach described in Section 3, to solve Eq. (8) we first solve the perturbed system $A'\mathbf{c}' = 6\mathbf{b}$, where A' and $\mathbf{c}'$ have been defined in Section 3. Then $\mathbf{c}$ can be obtained by updating $\mathbf{c}'$. It is clear that

$$T = A' + \begin{pmatrix} -b & & 1 \\ & & \\ & & \\ & 1 & \end{pmatrix}. \quad (9)$$

By Eq. (9) and $\mathbf{b}' = 6\mathbf{b}$, we obtain

$$T\mathbf{c}' = A'\mathbf{c}' + (c'_n - bc'_1)\mathbf{e}_1 + c'_1\mathbf{e}_n$$
$$= 6\mathbf{b} + (c'_n - bc'_1)\mathbf{e}_1 + c'_1\mathbf{e}_n. \quad (10)$$

Table 1. Time required when running on IBM-386 PC for cubic B-spline curve fitting.

n	Gaussian elimination	Our algorithm	Ratio
64	0.0027s	0.0022s	0.815
128	0.0055s	0.0043s	0.782
256	0.0111s	0.0083s	0.747
512	0.0223s	0.0164s	0.735
1024	0.0446s	0.0326s	0.731
2048	0.0894s	0.0651s	0.728

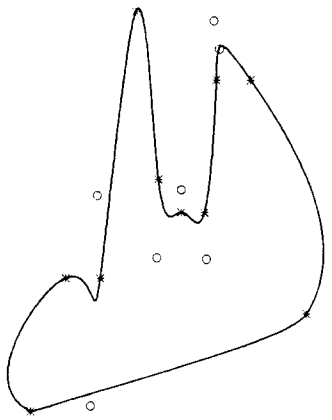

Fig. 2. An example of the closed cubic B-spline curve fitting.

Therefore, we have

$$\mathbf{c} = \mathbf{c}' - (1 - b)c_1'A^{-1}\mathbf{e}_1 - c_n'A^{-1}\mathbf{e}_n.$$

It can be easily verified that $T\mathbf{c} = 6\mathbf{b}$. By the similar arguments in computing $\mathbf{x}$ and $\mathbf{y}$, which are approximated solutions of $A^{-1}\mathbf{e}_1$ and $A^{-1}\mathbf{e}_n$ [see Eqs. (5) and (6)], the approximated solution of $\mathbf{c}$ can be solved as follows.

We have known that $\mathbf{x} = \frac{1}{b-1}(\underbrace{b, b^2, \ldots, b^p}_{p}, \underbrace{0, \ldots, 0}_{n-p})^t$ $\mathbf{y} = \frac{1}{b-1}(\underbrace{0, \ldots, 0}_{n-p}, \underbrace{b^p, \ldots, b^2, b}_{p})^t$. So we have

$$(b-1)T\mathbf{x} = (\underbrace{4b + b^2, 0, \ldots, 0, b^{p-1} + 4b^p}_{p}, \underbrace{b^p, 0, \ldots, 0, b}_{n-p})^t$$
$$= -\mathbf{e}_1 + b\mathbf{e}_n - b^{p+1}\mathbf{e}_p + b^p\mathbf{e}_{p+1}. \quad (11)$$

Similarly, we have

$$(b-1)T\mathbf{y} = b\mathbf{e}_1 - \mathbf{e}_n - b^{p+1}\mathbf{e}_{n+1-p} + b^p\mathbf{e}_{n-p}. \quad (12)$$

By Eqs. (11) and (12), it gives

$$(b-1)T(\mathbf{x} + b\mathbf{y}) = b^2 - 1)\mathbf{e}_1 - b^{p+1}\mathbf{e}_p + b^p\mathbf{e}_{p+1} - b^{p+2}\mathbf{e}_{n+1-p} + b^{p+1}\mathbf{e}_{n-p};$$
$$(b-1)T(b\mathbf{x} + \mathbf{y}) = (b^2 - 1)\mathbf{e}_n - b^{p+2}\mathbf{e}_p + b^{p+1}\mathbf{e}_{p+1} - b^{p+1}\mathbf{e}_{n+1-p} + b^p\mathbf{e}_{n-p}.$$

Let $\mathbf{v} = \frac{1}{b+1}(\mathbf{x} + b\mathbf{y})$ and $\mathbf{w} = \frac{1}{b+1}(b\mathbf{x} + \mathbf{y})$, then we have

$$T\mathbf{v} = \mathbf{e}_1 - \frac{b^p}{b^2-1}(b\mathbf{e}_p - \mathbf{e}_{p+1} + b^2\mathbf{e}_{n+1-p} - b\mathbf{e}_{n-p});$$

$$T\mathbf{w} = \mathbf{e}_n - \frac{b^p}{b^2-1}(b^2\mathbf{e}_p - b\mathbf{e}_{p+1} + b\mathbf{e}_{n+1-p} - \mathbf{e}_{n-p}).$$

Let $\mathbf{c} = \mathbf{c}' - (c_n' - bc_1')\mathbf{v} - c_1'\mathbf{w}$, then $T\mathbf{c} = T\mathbf{c}' - (c_n' - bc_1')T\mathbf{v} - c_1'T\mathbf{w}$. Furthermore, we have

$$T\mathbf{c} - 6\mathbf{b} = (c_n' - bc_1')\frac{b^p}{b^2-1}(b\mathbf{e}_p - \mathbf{e}_{p+1} + b^2\mathbf{e}_{n+1-p} - b\mathbf{e}_{n-p}) + c_1'\frac{b^p}{b^2-1}(b^2\mathbf{e}_p - b\mathbf{e}_{p+1} + b\mathbf{e}_{n+1-p} - \mathbf{e}_{n-p}) = \frac{b^p}{b^2-1}(c_n'b\mathbf{e}_p - c_n'\mathbf{e}_{p+1} + (c_n'b^2 + c_1'(b - b^3))\mathbf{e}_{n+1-p} - (c_n'b + c_1'(1 - b^2))\mathbf{e}_{n-p}).$$

Before discussing the bound of $\|T\mathbf{c} - 6\mathbf{b}\|$, we first need the following lemma.

Lemma 2. $\|\mathbf{c}'\| \le 3\|\mathbf{b}\|$.

Proof. The proof is similar to Lemma 1.

If p, $p+1$, $n+1-p$, $n-p$ are distinct, by Lemma 2, then it yields

$$\begin{aligned}\|T\mathbf{c} - 6\mathbf{b}\| &\le \left|\frac{b^p}{b^2-1}\right|\max(|bc_n'|, |c_n'|, |c_n'b^2 + c_1'(b - b^3)|, |(c_n'b + c_1'(1 - b^2)|)\\ &= \left|\frac{b^p}{b^2-1}\right|\max(|c_n'|, |(c_n'b + c_1'(1 - b^2)|)\\ &= \left|\frac{b^p}{b^2-1}\right|\max(3\|\mathbf{b}\|, 3\|\mathbf{b}\|(|b| + |1 - b^2|))\\ &= \left|\frac{b^p}{b^2-1}\right| 3\|\mathbf{b}\|(|b| + |1 - b^2|)\\ &= \frac{3\sqrt{3} - 4}{4\sqrt{3} - 6}|b|^p 3\|\mathbf{b}\| < 3.866|b|^p\|\mathbf{b}\|.\end{aligned}$$

For example, if $\|\mathbf{b}\| \le 1000$ and $p = 10$, then the residual is $\le 7.4 \times 10^{-3}$. Under this conditions, in our implementation, the residual for the solution $\mathbf{c}$ ($n > 10$) is of order 10^{-3}.

Therefore, the closed cubic B-spline curve fitting becomes a three-phase process, namely, performing Toeplitz factorization first, second solving $A'\mathbf{c}' = 6\mathbf{b}$ for $\mathbf{c}'$ and then computing

$$\mathbf{c} = \mathbf{c}' - (c'_n - bc'_1)\mathbf{v} - c'_1\mathbf{w}$$

$$= \mathbf{c}' - \frac{c'_n}{b+1}\mathbf{x} - \frac{bc'_n + (1-b^2)c'_1}{b+1}\mathbf{y}$$

$$= \mathbf{c}' - \frac{c'_n}{b^2-1}(\underbrace{b, b^2, \ldots, b^p}_{p}, \underbrace{0, \ldots, 0}_{n-p})^t$$

$$- \frac{bc'_n + (1-b^2)c'_1}{b^2-1}(\underbrace{0, \ldots, 0}_{n-p}, \underbrace{b^p, \ldots, b^2, b}_{p})^t,$$

which is performed in the update phase, the third phase, to obtain **c**. The corresponding three-phase algorithm can be designed in a similar way as in the open case, and the number of FP operations required is also about $5n$. It comes to a conclusion that the number of FP operations required in our algorithm is one-third as many as the one using Gaussian elimination. The **C** language code of the closed cubic B-spline curve fitting algorithm using Gaussian elimination and our three-phase approach are given in Appendix C and Appendix D, respectively.

Table 2 shows the performances of running our algorithm and the one using Gaussian elimination on IBM-386 PC.

It is observed that our three-phase approach for solving Eq. (8) is faster than Gaussian elimination method. The value of "ratio" is near to the theoretic value 1/3.

5. CONCLUSIONS

We have presented fast three-phase algorithms for open and closed cubic B-spline curve fittings. Our algorithms have been implemented in **C** language codes on IBM-386 PC to show the good performances when compared with Gaussian elimination methods. In fact, our result can be applied to design fast algorithms for solving many other curve fitting problems such as the quadratic B-spline curve fitting[14, 16]. However, our result cannot be extended to handle the nonuniform case[12] but how to speed up the computation for this case is our future research topic.

Previously, many methods were proposed for solving the tridiagonal near-Toeplitz systems. These methods are special *LU* factorization[10], cyclic reduction[9], reversed triangular factorization[5–7], and Toeplitz factorization with Sherman-Morrison formula[8]. The interested readers are suggested to consult the survey paper by Boisvert[3]. For solving the open as well as the closed cubic B-spline curve fitting problems, the number of FP operations required in our new algorithms is the same as the previous fastest ones such as the special *LU* factorization and reversed triangular factorization[3]. Pham[14] proposed a digital filter approach to solve the quadratic B-spline curve fitting problem, but his paper did not analyze the time complexity needed, error analysis, and the comparison with Gaussian elimination.

Table 2. Time required when running on IBM-386 PC for closed cubic B-spline curve fitting.

n	Gaussian elimination	Our algorithm	Ratio
64	0.0060s	0.0023s	0.383
128	0.0160s	0.0043s	0.269
256	0.0285s	0.0084s	0.295
512	0.0534s	0.0164s	0.307
1024	0.1030s	0.0326s	0.316
2048	0.2032s	0.0649s	0.319

Acknowledgements—This research was supported in part by the National Science Council of R. O. C. under Grant NSC82-0415-E011-180.

REFERENCES

1. B. A. Barsky, End conditions and boundary conditions for uniform B-spline curve and surface representations. *Comp. Industry* **3** (1/2), 17–29 (1982).
2. R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Section 4.2: Uniform Cubic B-splines, Morgan Kaufmann, San Mateo, CA (1987).
3. R. F. Boisvert, Algorithms for special tridiagonal systems. *SIAM J. Sci. Stat. Comp.* **12**, 423–442 (1991).
4. G. M. Chaikin, An algorithm for high-speed curve generation. *Comp. Graph. Image Proc.*, **3**, 346–349 (1974).
5. D. J. Evans and C. D. V. Forrington, Note on the solution of certain tri-diagonal systems of linear equations. *Comp. J.* **5**, 327–328 (1963).
6. D. J. Evans, An algorithm for the solution of certain tridiagonal systems of linear equations. *Comp. J.* **15**, 356–359 (1972).
7. D. J. Evans, On the the solution of certain Toeplitz tridiagonal linear systems. *SIAM J. Numer. Anal.* **17**, 675–680 (1980).
8. D. Fischer, G. Golub, O. Hald, C. Levia, and O. Winlund, On Fourier-Toeplitz methods for separable elliptic problems. *Math. Comp.* **28** (126), 349–368 (1974).
9. R. W. Hockney, A fast direct solution of Poisson's equation using Fourier analysis. *J. ACM.* **12**, 95–113 (1965).
10. M. A. Malcolm and J. Palmer, A fast method for solving a class of tridiagonal linear systems. *Comm. ACM.* **17**, 14–17 (1974).
11. J. D. Foley and Van Dam, *Fundamentals of Interative Computer Graphics*, Addison-Wesley, Reading, MA (1982).
12. J. Hoschek and D. Lasser, *Fundamentals of Computer Aided Geometric Design*, A. K. Peters, Ltd. (1993).
13. T. Pavlidis, Curve fitting as a pattern recognition problem. In *Proceedings of the 6th International Conference on Pattern Recognition*, Munich, Germany, IEEE Computer Society Press, 853–859 (1982).
14. B. Pham, Quadratic B-splines for automatic curve and surface fitting. *Comp. & Graph.* **13**, 471–475 (1989).
15. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Academic Press, New York (1982).
16. R. F. Riesenfeld, On Chaikin's algorithm. *Comp. Graph. Image Proc.* **4**, 304–310 (1974).

APPENDIX A

```
/*Gaussian Elimination for Open Cubic B-spline
Curve Fitting*/
#include <stdio.h>
#include <stdlib.h>
main()
{ float res,temp,a[5000],b[5000],c[5000];
  int i,n;
  printf(''Gaussian Elimination for Open Cubic
  B-spline Curve Fitting\n");
  printf(''INPUT N:'');/*N:number of the given
  points*/
  scanf(''%d'',&n);
```

```
/*Generate Random Given Points*/
  for  (i=1; i<=n; i++)  {a[i]=rand()  %
  1000;b[i]=a[i];}
  for (i=1;i<=n;i++) a[i]*=6.0;
/*Forward-elimination*/
  c[1]=0.2;a[1]*=0.2;
  for (i=2;i<=n-1;i++) {
    c[i]=1/(4.0-c[i-1]);
    a[i]=(a[i]-a[i-1])*c[i];
  }
  a[n]=(a[n]-a[n-1])/(5.0-c[n-1]);
/*Backward-substitution*/
  for (i=n-1;i>=1;i--) a[i]-=c[i]*a[i+1];
/*Check the Residual of Solution*/
  res=5.0*a[1]+a[2]-6.0*b[1]; /*res is used to
  save the residual*/
  if (res<0) res=-res;
  for (i=2;i<=n-1;i++) {
    temp=a[i-1]+4.0*a[i]+a[i+1]-6.0*b[i];
    if (temp<0) temp=-temp;
    if (res<temp) res=temp;
  }
  temp=a[n-1]+5.0*a[n]-6.0*b[n];
  if (temp<0) temp=-temp;
  if (res<temp) res=temp;
  printf(''\nThe Residual=%10.7f\n",res);
}
```

APPENDIX B

```
/*Our Method for Open Cubic B-spline Curve Fit-
ting*/
#include <stdio.h>
#include <stdlib.h>
#define alpha -0.2679492 /
*sqrt(3)-2=-0.2679492*/
#define p 10
main()
{ float temp,res,x[20],a[5000],b[5000];
  int i,n;
  printf(''Our Method for Open Cubic B-spline
  Curve Fitting\n");
  printf(''INPUT N:''); /*N:number of the given
  points*/
  scanf(''%d'',&n);
/*Generate Array X for Updation*/
  for (i=1,x[0]=1.0,1t=p;i++)
  x[i]=alpha*x[i-1];
/*Generate Random Given Points*/
  for (i=1;i<=n;i++) {a[i]=rand() %
  1000;b[i]=a[i];}
/*Solving L'Y=6B*/
  for (i=1;i<=n;i++) a[i]*=6.0;
  for (i=2;i<=n;i++) a[i]+=alpha*a[i-1];
/*Solving U'C=Y*/
  a[n]*=(-alpha);
  for (i=n-1;i>=1;i--)
  a[i]=alpha*(a[i+1]-a[i]);
/*Updation*/
  temp=a[1];
  for (i=1;i<=p;i++) a[i]+=x[i]*temp;
  temp=a[n]/(alpha-1);
  for (i=1;i<=p;i++) a[n+1-i]-=x[i]*temp;
/*Check the Residual of Solution*/
  res=5.0*a[1]+a[2]-6.0*b[1]; /*res is used to
  save the residual*/
  if (res<0) res=-res;
  for (i=2;i<=n-1;i++) {
    temp=a[i-1]+4.0*a[i]+a[i+1]-6.0*b[i];
    if (temp<0) temp=-temp;
    if (res<temp) res=temp;
  }
  temp=a[n-1]+5.0*a[n]-6.0*b[n];
  if (temp<0) temp=-temp;
  if (res<temp) res=temp;
  printf(''\nThe Residual=%10.7f\n'',res);
}
```

APPENDIX C

```
/*Gaussian Elimination for Closed Cubic B-
spline Curve Fitting*/
#include <stdio.h>
#include <stdlib.h>
main()
{ float res,temp,a[3000],b[3000],c[3000],
  d[3000];
  int i,n;
  printf(''Gaussian Elimination for Closed Cu-
  bic B-spline Curve Fitting\n'');
  printf(''INPUT N:''); /*N:number of the given
  points*/
  scanf(''%d'',&n);
/*Generate Random Given Points*/
  for (i=1;i<=n;i++) {a[i]=rand()
  %1000;b[i]=a[i];}
  for (i=1;i<=n;i++) a[i]*=6.0;
/*Forward-elimination*/
  c[1]=0.25;d[1]=0.25;a[1]*=0.25;
  for (i=2;i<=n-2;i++){
    c[i]=1/(4.0-c[i-1]);d[i]=-d[i-1]*c[i];
    a[i]=(a[i]-a[i-1])*c[i];
  }
  c[n-1]=0;
  temp=1.0/(4-c[n-2]);
    d[n-1]=(1.0-d[n-2])*temp;
    a[n-1]=(a[n-1]-a[n-2])*temp;
  temp=1.0/(4.0-d[n-1]);
  a[n]=(a[n]-a[n-1])*temp;
  d[n]=1.0;
  for (i=1;i<=n-2;i++) {
    a[n]=a[n]-temp*a[i];
    d[n]=d[n]-temp*d[i];
    temp=-c[i]*temp;
  }
  a[n]=(a[n]-temp*a[n-1])/
  (d[n]-temp*d[n-1]);
/*Backward-substitution*/
  for (i=n-1;i>=1;i--)
  a[i]=a[i]-c[i]*a[i+1]-d[i]*a[n];
/*Check the Residual of Solution*/
  res=4.0*a[1]+a[2]+a[n]-6.0*b[1];
  if (res<0) res=-res;
  for (i=2;i<=n-1;i++) {
    temp=a[i-1]+4.0*a[i]+a[i+1]-6.0*b[i];
    if (temp<0) temp=-temp;
    if (res<temp) res=temp;
  }
  temp=a[1]+a[n-1]+4.0*a[n]-6.0*b[n];
  if (temp<0) temp=-temp;
  if (res<temp) res=temp;
  printf(''\nThe Residual=%10.7f\n'',res);
}
```

APPENDIX D

```
/*Our Method for Closed Cubic B-spline Curve
Fitting*/
#include <stdio.h>
#include <stdlib.h>
#define p 10
#define alpha -0.2679492
main()
{ float temp1,temp2,temp,res,x[20],a[5000],
  b[5000];
  int i,n;
  printf(''Our Method for Closed Cubic B-spline
  Curve Fitting\n'');
  printf(''INPUT N:''); /*N:number of the given
  points*/
```

```
  scanf(''%d'',&n);
/*Generate Array X for Updation*/
  for (i=1,x[0]=1.0;i<=p;i++)
  x[i]=alpha*x[i-1];
/*Generate Random Given Points*/
  for (i=1;i<=n;i++) {a[i]=rand() %
  1000;b[i]=a[i];}
/*Solving L'Y=6B*/
  for (i=1;i<=n;i++) a[i]*=6.0;
  for (i=2;i<=n;i++) a[i]+=alpha*a[i-1];
/*Solving U'C=Y*/
  a[n]*=(-alpha);
  for (i=n-1;i>=1;i--) a[i]=alpha*
  (a[i+1]-a[i]);
/*Updation*/
  temp1=a[n]/(alpha*alpha-1);
  temp2=-a[1]+a[n]*alpha/(alpha*alpha-1);
  for (i=1;i<=p;i++) a[i]-=x[i]*temp1;
  for (i=1;i<=p;i++) a[n+1-i]-=x[i]*temp2;
/*Check the Residual of Solution*/
  res=4.0*a[1]+a[2]+a[n]-6.0*b[1]; /*res is
  used to save the residual*/
  if (res<0) res=-res;
  for (i=2;i<=n-1;i++){
    temp=a[i-1]+4.0*a[i]+a[i+1]-6.0*b[i];
    if (temp<0) temp=-temp;
    if (res<temp) res=temp;
  }
  temp=a[1]+a[n-1]+4.0*a[n]-6.0*b[n];
  if (temp<0) temp=-temp;
  if (res<temp) res=temp;
  printf(''\nThe Residual=%10.7f\n'',res);
}
```