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[7].

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, \( B_i = (b_{i1}, b_{i2}, \ldots, b_{in}) \) for \( 1 \leq i \leq n \). According to [2], for an uniform cubic B-spline curve, each given point can be expressed by a weighted average of three control points:

\[
B_i = \frac{1}{4}(C_{i-1} + 4C_i + C_{i+1}), \quad 1 \leq i \leq n,
\]

where \( C_i = (c_{i1}, c_{i2}, \ldots, c_{in}) \). 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 \( b = (b_1, b_2, \ldots, b_n)' = (b_1, b_2') = (b_3', b_3') = \cdots = (b_n') = (c_{11}, c_{12}, \ldots, c_{1n})' \) 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

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

First, it takes \( n \) FP operations to perform the multiplication \( 6 b \). 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, \(C_n\), of the \(n + 2\) defining polygon vertices. Letting \(P_\text{i}(t)\) be the position vectors along the \(i\)th piecewise cubic curve as a function of the parameter \(t\), the \(i\)th cubic B-spline curve segment is given by

\[
P_i(t) = \sum_{j=-1}^{2} C_{i+j}N_j(t) \quad \text{for} \quad 1 \leq i \leq n - 1 \tag{2}
\]

and \(0 \leq t < 1\),

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:

\[
N_0(t) = \frac{-t^3 + 3t^2 - 3t + 1}{6}; \quad N_1(t) = \frac{3t^3 - 6t^2 + 4}{6}; \quad N_2(t) = \frac{-3t^3 + 3t^2 + 3t + 1}{6}; \quad N_3(t) = \frac{t^3}{6}.
\]

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 \(A\),

\[
A' = L'U' = \begin{pmatrix}
1 & 1 & 1 \\
4 & 1 & \\
4 & 1 & 1 \\
\end{pmatrix}
\]

where

\[
L' = \begin{pmatrix}
1 & 1 & \\
-1 & 1 & \\
& -1 & 1 \\
\end{pmatrix}
\]

and

\[
U' = \begin{pmatrix}
1 & \\
& 1 & \\
& & 1 & \\
\end{pmatrix}
\]

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.
\[ A = A' + \left( \begin{array}{c} 1 - b \\ 0 \end{array} \right) . \quad (3) \]

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

\[
\begin{align*}
\text{for } i = 1 \text{ to } n \text{ do } b_i' & = 6b_i' \\
\text{for } i = 2 \text{ to } n \text{ do } c_i' & = b_i' + b\ast c_{i-1}' \\
\text{for } i = n-1 \text{ downto } 1 \text{ do } c_i' & = b\ast (c_{i+1}' - b_i')
\end{align*}
\]

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 \( b' = 6b \), we obtain

\[ Ac = A'c' + (1 - b)c_e + c_e' = 6b + (1 - b)c_e + c_e', \quad (4) \]

where \( c_e = (0, 0, \ldots, 0)' \) and \( c_e' = (0, 0, \ldots, 0, 1)' \).

Hence, it yields

\[ c = c' - (1 - b)c_e' - c_{n-1}'c_{n-1}e_e. \]

It can be easily verified that \( Ac = 6b \).

Solving the recurrence relations: \( x_{i-1} + 4x_i + x_{i+1} = 0 \) for \( 2 \leq i \leq n - 1 \), we obtain \( x_i = \beta b^i \left( -\frac{1}{2} \right) \) for some constants \( \beta \) and \( \gamma \). 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 \( x_i = \alpha b^i \). By the first equation \( 5x_1 + x_2 = 1 \), we have \( x_i = \frac{b^i}{b - 1} \) for \( 1 \leq i \leq n \). When \( i \) is sufficiently large, say, \( i = n \), \( x_i \approx 0 \) and the last equation \( x_{n-1} + 5x_n = 0 \).

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

\[
x_i = \begin{cases} 
\frac{b^i}{b - 1} & \text{if } 1 \leq i \leq p (p: \text{a small integer}) \\
0 & \text{if } p + 1 \leq i \leq n
\end{cases}
\]

and \( x = (x_1, x_2, \ldots, x_n)' \), then we have

\[ Ax = e_1 - \frac{b^{p+1}}{b - 1} e_p + \frac{b^p}{b - 1} e_{n-1}. \quad (5) \]

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

\[
y_i = \begin{cases} 
\frac{b^{n+1-i}}{b - 1} & \text{if } n - p + 1 \leq i \leq n \\
0 & \text{if } 1 \leq i \leq n - p
\end{cases}
\]

and \( y = (y_1, y_2, \ldots, y_n)' \), then we have

\[ Ay = e_1 - \frac{b^{n+1}}{b - 1} e_{n-1} + \frac{b^p}{b - 1} e_{n-p}. \quad (6) \]

Let

\[
c = c' - c_1(1 - b)x - c_2y
\]

\[ = c' + c_1b b', b', b', \ldots, 0, \ldots, 0, 0 (n-p)
\]

\[ + \frac{c_{n-p}}{b - 1} (0, 0, \ldots, 0, b, b', b'). \]

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

\[ Ac = 6b - (-c_1b^{p+1}e_p + c_1b^p e_{n-p}) \]

\[ + \frac{c_{n-p}}{b - 1} b^{n+1} e_{n-1} + \frac{c_{n-p}}{b - 1} b^p e_{n-p}. \quad (7) \]

Under a satisfactory residual requirement, say, \( \|Ac - 6b\| \) 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.** \( \|c'\| \leq 3\|\| \), where \( \|\| = \max_{i: i \in \mathbb{N}} (\|x_i\|) \).

**Proof.** Suppose \( \|c'\| > 3\|\| \). If \( \|c_i'\| = \|c\| \), for some \( i \) \( (2 \leq i \leq n - 1) \), we have \( c_{i+1}' + 4c_i' + c_{i-1}' = 6b_i' \). Thus it gives \( 4\|c_i\| \leq \|6b_i\| + \|c_{i-1}'\| + \|c_{i+1}'\|, \text{ i.e.,} \)

\[ \|6b_i\| \geq 4\|c_i\| - \|c_{i-1}'\| - \|c_{i+1}'\| \]

\[ \geq 4\|c\| - \|\| - \|\| = 2\|\| > 6\|b\|. \]

It is a contradiction.

If \( \|c_i'\| = \|\| \) for \( i = 1 \), we have \( \left(-\frac{1}{b}\right)c_1' = 6b_1 - c_2' \),

\[ \left(-\frac{1}{b}\right)c_1' \leq \|c\| \]

and \( \left(-\frac{1}{b}\right)c_1' \leq \|6b_1\| + \|c_2\| \). Thus it gives

\[ \|6b_1\| \geq \left| \left(-\frac{1}{b}\right)c_1' \right| - \|c_2\| \geq \left| \left(-\frac{1}{b}\right)\|\| - \|\| \right| \]

\[ = (1 + \sqrt{3})\|\| > 3(1 + \sqrt{3})\|b\|. \]

It is a contradiction.

If \( \|c_i'\| = \|\| \) for \( i = n \), we have \( 4c_n' = 6b_n - c_{n-1}' \),

\[ 4\|c_n\| \leq \|6b_n\| + \|c_{n-1}'\| \]

and \( 4\|c_n\| \leq \|6b_n\| + \|c_{n-1}'\| \). Thus it gives
\[ |6b_n| \geq 4c_n^2 - |c_{n-1}| \geq 4|c'| - |c| \]
\[ = 3|c'| > 9\|b\|. \]

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 our three-phase algorithm has the residual

\[ \|4c - 6b\| \leq \max\left( \left| c_1^t b^{p+1} \right|, \left| c_1^t b^p \right|, \left| c_2^t b^{p+1} \right|, \left| c_2^t b^p \right| \right) \]
\[ = \max\left( \left| c_1^t b^p \right|, \left| c_2^t b^p \right| \right) \]
\[ \leq \max\left( 3\|b\| \left| b^p \right|, 3\|b\| \left| b^p \right| \right) = 3\|b\| \left| b^p \right| \]

That is, we have

\[ \|4c - 6b\| \leq 3\|b\| \left| b^p \right|. \]

For example, if \( \|b\| \leq 1000 \) and \( p = 10 \), then the residual \( \leq 5.7 \times 10^{-3} \). Under these conditions, in our implementation, the residual for the solution \( \varepsilon(n > 10) \) 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.”

<table>
<thead>
<tr>
<th>( n )</th>
<th>Gaussian elimination</th>
<th>Our algorithm</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.0027s</td>
<td>0.0022s</td>
<td>0.815</td>
</tr>
<tr>
<td>128</td>
<td>0.0055s</td>
<td>0.0043s</td>
<td>0.782</td>
</tr>
<tr>
<td>256</td>
<td>0.0111s</td>
<td>0.0083s</td>
<td>0.747</td>
</tr>
<tr>
<td>512</td>
<td>0.0223s</td>
<td>0.0164s</td>
<td>0.735</td>
</tr>
<tr>
<td>1024</td>
<td>0.0446s</td>
<td>0.0326s</td>
<td>0.731</td>
</tr>
<tr>
<td>2048</td>
<td>0.0894s</td>
<td>0.0651s</td>
<td>0.728</td>
</tr>
</tbody>
</table>

For 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 \( Ac' = 6b \), where \( A' \) and \( c' \) have been defined in Section 3. Then \( c \) can be obtained by updating \( c' \). It is clear that

\[ T = A' + \begin{pmatrix} -b & 1 \\ 1 & \end{pmatrix} \]

By Eq. (9) and \( b' = 6b \), we obtain

\[ Tc' = A'c' + (c_n^e - bc')e_1 + c'c_e \]
\[ = 6b + (c_n^e - bc')e_1 + c'c_e. \]
Let $v = \frac{1}{b + 1} (x + by)$ and $w = \frac{1}{b + 1} (bx + y)$, then we have

$$Tv = e_1 - \frac{b^p}{b^2 - 1} (be_p - e_{p+1} + b^2 e_{n+1-p} - be_{n-p});$$

$$T w = e_n - \frac{b^p}{b^2 - 1} (b^2 e_p - be_{p+1} + be_{n+1-p} - e_{n-p}).$$

Let $c = e' - (c' - bc') w - c' w$, then $Tc = Tc' - (c' - bc') Tv - c' Tw$. Furthermore, we have

$$Tc - 6b = (c' - bc') - \frac{b^p}{b^2 - 1} (be_p - e_{p+1})$$

$$+ b^2 e_{n+1-p} - be_{n-p}) + c' - \frac{b^p}{b^2 - 1} (b^2 e_p - be_{p+1})$$

$$+ be_{n+1-p} - e_{n-p}) = \frac{b^p}{b^2 - 1} (c' - bc' - c') e_{n-p}$$

$$+ (c' b^2 + c'(b - b^3)) e_{n+1-p} - (c' b + c'(b - b^2)) e_{n-p}. $$

Before discussing the bound of $\|Tc - 6b\|$, we first need the following lemma.

**Lemma 2.** $|c| \leq 3|b|$. 

**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

$$\|Tc - 6b\| \leq \frac{b^p}{b^2 - 1} \max(|bc' |, |c' |, |c' b^2)$$

$$+ c'(b - b^3), \|c' b + c'(1 - b^2)\|)$$

$$= \frac{b^p}{b^2 - 1} \max(|c' |, \|c' b + c'(1 - b^2)\|)$$

$$= \frac{b^p}{b^2 - 1} \max(3|b|, 3|b|) \|b\|(|b| + |1 - b^2|)$$

$$= \frac{b^p}{b^2 - 1} \max(3|b|, 3|b|) \|b\|(|b| + |1 - b^2|)$$

$$\leq \frac{1}{4} |b|^3 |b| < 3.866 |b|^3 |b|.$$

For example, if $|b| \leq 1000$ and $p = 10$, then the residual is $\leq 7.4 \times 10^{-3}$. Under this conditions, in our implementation, the residual for the solution $c$ ($n > 10$) 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 $A'c' = 6b$ for $c'$ and then computing...
\[ c = c' - (c_n - bc')v - c'w \\
= c' - \frac{c_n}{b+1} + \frac{bc_n + (1 - b^2)c'_i}{b+1} \gamma \\
= c' - \frac{c_n}{b^2 - 1} \left( \frac{h}{\beta_v^2}, \ldots, \frac{h}{\beta_v^2}, 0, \ldots, 0 \right)^t \\
- \frac{bc_n + (1 - b^2)c'_i}{b^2 - 1} \left( 0, \ldots, 0, h \beta_v^2, h \beta_v^2, b \beta_v^2, b \beta_v^2, \right) \]

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.

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).

APPENDIX A

```c
#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("INTEGER N\n");
  scanf("%d", &n);
-offs"
```
A fast algorithm for cubic B-spline curve fitting

```c
/*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]=a[i+1];

/*Forward-elimination*/
c[i]=0.2; for (i=1; i<=n; i++) a[i]=b[i];
for (i=2; i<=n; i++)
    a[i]-=alpha*a[i-1];
    c[i]-=(c[i-1]+4.0*a[i-1]-6.0*b[i];
if (temp<0) temp=-temp;
if (res<temp) res=temp;

/*Backward-substitution*/
for (i=n-1; i>=1; i--)
    a[i]-=(a[i]-a[i+1]);
    c[i]-=c[i-1]/(4.0-c[i-1]);
    d[i]-=d[i]*c[i];
    a[i]+-(a[i]-a[i+1])/(5.0-c[i-1]);
    c[i]-=(c[i-1]-a[i])/c[i];

/*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; 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;

/*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; 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;

APPENDIX B
/*Our Method for Open Cubic B-spline Curve Fitting*/
#include <stdio.h>
#include <stdlib.h>
#define alpha -0.2679492
#define p i0
main()
    float temp,res,x[20],a[5000],b[5000];
    int i,n;
    printf(''
Our Method for Open Cubic B-spline
Curve Fitting
'');
    printf(''
INPUT N:'');/*N: number of the given
    points*/
    scanf(''
%d
'',&n);
    for (i=1; i<n+1; i++) a[i]=rand() % 1000;
    b[i]=a[i];
    for (i=1; i<n; i++)
        a[i]=a[i+1];
    for (i=2; i<=n; i++)
        a[i]-=alpha*a[i-1];
    for (i=2; i<=n; 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(''
The Residual=%10.7f
'',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
'');
    printf(''
INPUT N:'');/*N: number of the given
    points*/
    scanf(''
%d
'',&n);
    for (i=1; i<n; i++)
        a[i]=rand() % 1000;
        b[i]=a[i];
        for (i=1; i<n; i++)
            a[i]=a[i+1];
            c[i]=0.25;
            d[i]=0.25;
            a[i]-=0.25;
            for (i=2; i<=n-2; i++)
                c[i]-=(c[i-1]+4.0*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(''
The Residual=%10.7f
'',res);

APPENDIX D
/*Our Method for Closed Cubic B-spline Curve Fitting*/
#include <stdio.h>
#include <stdlib.h>
#define alpha -0.2679492
#define p i0
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
'');
    printf(''
INPUT N:'');/*N: number of the given
    points*/
```
```c
scanf("%d", &n);
/*Generate Array X for Updation*/
for (i=1; 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*/
  templ = a[n] / (alpha * alpha - 1);
  temp2 = -a[1] + a[n] * alpha / (alpha * alpha - 1);
for (i=1; i<=p; i++) a[i] -= x[i] * templ;
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);
```