Computing 36, 229–238 (1986)

Computing
© by Springer-Verlag 1986

Short Communications / Kurze Mitteilungen

Comments on Some B-Spline Algorithms*

E. T. Y. Lee, Seattle

Received April 1, 1985

Abstract — Zusammenfassung

Comments on Some B-Spline Algorithms. Two algorithms appeared (Lee [6] and Boehm [1]), both in this journal, for the computation of derivatives of a *B*-spline series that are faster than the repeated application of de Boor's algorithm to the derived series. Here we report on some test examples which show, however, that both these are less stable than the repeated de Boor.

AMS Subject Classifications: 65-04, 65D07.
Key words: *B*-splines, derivatives, algorithms.

Bemerkungen zu einigen B-Spline-Algorithmen. In dieser Zeitschrift sind zwei Algorithmen zur Berechnung von Ableitungen einer *B*-Spline-Reihe erschienen (Lee [6] und Boehm [1]), die weniger Aufwand benötigen als die wiederholte Anwendung des Algorithmus von de Boor auf die abgeleitete Reihe. Wir berichten hier über Testrechnungen, die zeigen, daß diese Algorithmen weniger stabil sind als die wiederholte Anwendung des de Boor-Algorithmus.

1. Introduction

A standard way of evaluating a *B*-spline expansion of order k,

$$F(t)=\sum A_i B_{i,k}(t),$$

at a value $t=x$ is by the de Boor algorithm (first algorithm in [3]), forming convex combinations of the relevant coefficients A_i, (of the form $aX+bY$, $a+b=1$, a,b linear functions of x), repeatedly $k-1$ times. The derivative of F has coefficients obtained from those of F by forming difference quotients (of the form $(X-Y)/c$, c a constant), and its value at x is then computed as before. We refer to this method of finding all derivatives the Full de Boor algorithm, following [1]. It (and other algorithms) can be represented pictorially, as in [7], [1], on the tetrahedron in Fig. 1. We visualize the coefficients A_i as being located on the edge AB, those obtained by difference quotients as located on $A'B'$, etc. The repeated de Boor operations take AB to C, $A'B'$ to C', etc; and $C, C', \ldots, D$ represent the value, derivative, ..., and $(k-1)$-th derivative of F at x.

* To the memory of B. Tiger.

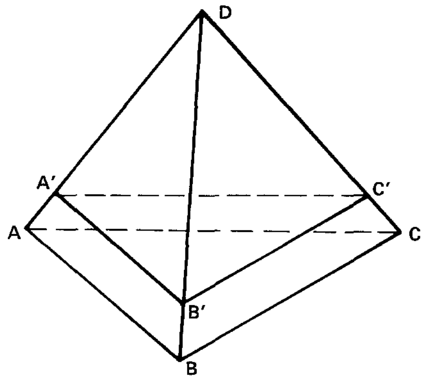

Fig. 1

A faster method in computing derivatives is obtained in [6], by noticing that while doing the de Boor operations AB to C, the intermediate values located on the edge AC (or BC) can be differenced to yield the derivatives. If x is nearer to the right end point of the knot interval, we use the values on AC, otherwise we use BC. Climbing up the face say from BC through $B'C'$, ..., we obtain the derivatives at $C', C'', \ldots, D$. The savings over the Full de Boor algorithm would be the convex combinations on the levels $A'B'C'$, $A''B''C''$, etc.

Recently Boehm [1] has devised an even faster algorithm for computing all derivatives of F. His method first finds the B-coefficients of the derivatives, i.e., the values on the face ABD. Then from the values on either AD or BD, one moves down the face DAC or DBC, obtaining $D, \ldots, C', C$, in that order, by operations of the type $X+cY$ (c a linear function of x. We may call this a Horner type operation). The number of steps involved is the same as that in Lee [6], the savings being due to the fact that the de Boor operations on ABC is slightly more expensive than the Horner operations in Boehm, on DAC (or DBC).

In this note we wish to report on some numerical experiments, which unfortunately show that both Boehm's algorithm and our previous one [6] are numerically less stable than the Full de Boor algorithm. Specifically, inaccuracies may occur in the higher derivatives by the method of Lee, whereas by Boehm's method, they occur in the value and lower derivatives.

2. Boehm's Algorithm

Let the knots be (t_i), the degree be $m=k-1$. With a suitable re-indexing, we may always consider x in the knot interval $I=[t_m, t_{m+1}]$. The relevant knots (for $m>0$) are then

$$t_1, \ldots, t_{2m},$$

(t_0, t_{2m+1} do not contribute to the values of F on I, for $m>0$); and the relevant B-coefficients are

$$A_0, \ldots, A_m.$$

Since Boehm did not present his algorithm in a form ready for coding, we will do so as our basis for comparison. In fact we give two versions, Boehm 1 and Boehm 2, according as whether we use the face ADC or BDC. The outputs are $Q_0, \ldots, Q_m$, with

$$Q_i = ((m-i)!/m!)\, D^i F(x),$$

the scaled derivatives.

BOEHM 1

```
     If $m=0$, $Q_0 := A_0$ and exit.
     For $i=0$ up to $m$
         $Q_i := A_i$.
     For $i=1$ up to $m$
         $a_i := x - t_i$.
     For $j=1$ up to $m$
     for $i=j$ up to $m$
         $d_{i,j} := t_{m+1+i-j} - t_i$.
(i)  For $j=1$ up to $m$
     for $i=m$ down to $j$
         $Q_i := (Q_i - Q_{i-1})/d_{i,j}$.
(ii) For $j=1$ up to $m$
     for $i=0$ up to $m-j$
         $Q_i := Q_i + a_{i+j} Q_{i+1}$.
```

BOEHM 2

```
     If $m=0$, $Q_0 := A_0$ and exit.
     For $i=0$ up to $m$
         $Q_i := A_{m-i}$.
     For $i=1$ up to $m$
         $a_i := t_{2m+1-i} - x$.
     For $j=1$ up to $m$
     for $i=j$ up to $m$
         $d_{i,j} := t_{2m+1-i} - t_{m-i+j}$.
(i)  For $j=1$ up to $m$
     for $i=m$ down to $j$
         $Q_i := (Q_{i-1} - Q_i)/d_{i,j}$.
(ii) For $j=1$ up to $m$
     for $i=0$ up to $m-j$
         $Q_i := Q_i - a_{i+j} Q_{i+1}$.
```

Some words about speed. We use the notation in [1]: $\sum m = m(m+1)/2$ and $\sum\sum m = m(m+1)(m+2)/6$. The cost of the Full de Boor algorithm is of the order $\sum\sum m$.

If, as in Boehm [1], we regard A_i as vectors of N dimensions, $N \gg 1$, (i.e., we consider computing a large number of scalar splines with the same knot distribution), then one may ignore the preprocessing cost and consider the cost as that spent in the main loops (i) and (ii), which is easily seen as

$$(2\,\text{add}+1\,\text{mult}+1\,\text{div})\sum m.$$

For comparison, one must slightly alter the algorithm in [6] to yield as above the scaled derivatives. The operation count depends on how one writes the de Boor operations in the base ABC. If one writes it as $aX+bY$, with a,b precomputed outside the main loops, then the cost in the main loops is

$$(2\,\text{add}+2\,\text{mult}+1\,\text{div})\sum m.$$

Actual tests (on an IBM 4341) show that, with all derivatives computed, Boehm is faster than Lee by about 20%, hadrly the "more than twice" as stated in [2]. Moreover, if one writes $aX+bY$ as $Y+a(X-Y)$, then the main loops cost only

$$(3\,\text{add}+1\,\text{mult}+1\,\text{div})\sum m;$$

and tests show Boehm faster than this version by only about 10%. However, we do not advocate this version since the sacrifice of convex combinations might lead to slight inaccuracies.

3. Stability

In Boehm, all derivatives, whether one needs them or not, are computed before the function value. The main reason for computing all derivatives at a point seems to be for the conversion of the given B-spline represention to its piecewise polynomial form, under the assumption that a large number of evaluations per knot interval is expected (so that subsequent computations can make use of the Horner's method). If so, the speed of conversion is clearly not an issue. Rather, it is important that the Taylor coefficients of each polynomial piece be obtained as accurately as possible, because of the potential numerical problems with the power basis (viz., large condition number).

Without going into intricate error analysis, we will compare the numerical stability of these algorithms simply by running test cases in both single and double precision: any discrepancy between the two would be an indication of the effect of roundoff errors. FORTRAN versions of the algorithms have been run on an IBM 4341, and some examples are listed below. The numbers are values of the scaled derivatives $Q_0, Q_1, \ldots$ as defined in section 2.

Example 1 (Inaccuracy of Boehm 1): $m=3$, $t_1, \ldots, t_6=0$, 10, 20, 30, 40, 50, $A_0, \ldots, A_3=100$, 0, 0, 0 (an individual B-spline basis function). $I=[20, 30]$, $x=30$. The data below show both the single and double precision results. Boehm 1 is clearly less satisfactory.

```
      BOEHM1  :

 0 :        -0.739098E-05           -0.17208456881690E-14
 1 :        -0.357628E-06           -0.83266726846887E-16
 2 :         0.596046E-07            0.13877787807814E-16
 3 :        -0.166667E-01           -0.16666666666667E-01
```

```
     BOEHM2 :

0 :        0.000000E+00              0.00000000000000E+00
1 :        0.000000E+00              0.00000000000000E+00
2 :        0.000000E+00              0.00000000000000E+00
3 :       -0.166667E-01             -0.16666666666667E-01

     LEE :

0 :        0.000000E+00              0.00000000000000E+00
1 :        0.000000E+00              0.00000000000000E+00
2 :        0.000000E+00              0.00000000000000E+00
3 :       -0.166667E-01             -0.16666666666667E-01

     FULL DE BOOR :

0 :        0.000000E+00              0.00000000000000E+00
1 :        0.000000E+00              0.00000000000000E+00
2 :        0.000000E+00              0.00000000000000E+00
3 :       -0.166667E-01             -0.16666666666667E-01
```

The final values Q_i's delivered in loop (ii) in Boehm 1 involve Horner operations of the form $X+(x-t_m)\,Y$; and in Boehm 2, those of the form $X-(t_{m+1}-x)\,Y$. It would thus appear that one might make a choice, using Boehm 2 if x is nearer the right end point. Such a strategy, however, does not work in general, as the next example shows.

Example 2 (Inaccuracy of Boehm 2): $m=3$, $t_1, \ldots, t_6=0$, 1, 2, 3, 400, 401, $A_0, \ldots, A_3=0$, 0, 0, 100. $I=[2,3]$, $x=3$. Here Boehm 2 is more prone to round-off errors.

```
     BOEHM1 :

0 :        0.629715E-03              0.62971499099508E-03
1 :        0.629715E-03              0.62971499099508E-03
2 :        0.629715E-03              0.62971499099508E-03
3 :        0.629715E-03              0.62971499099508E-03

     BOEHM2 :

0 :        0.617146E-03              0.62971499099444E-03
1 :        0.629783E-03              0.62971499099508E-03
2 :        0.629715E-03              0.62971499099508E-03
3 :        0.629715E-03              0.62971499099508E-03

     LEE :

0 :        0.629715E-03              0.62971499099508E-03
1 :        0.629715E-03              0.62971499099508E-03
2 :        0.629715E-03              0.62971499099508E-03
3 :        0.629715E-03              0.62971499099508E-03

     FULL DE BOOR :

0 :        0.629715E-03              0.62971499099508E-03
1 :        0.629715E-03              0.62971499099508E-03
2 :        0.629715E-03              0.62971499099508E-03
3 :        0.629715E-03              0.62971499099508E-03
```

Example 3 (Failure of Boehm at a zero): $m=5$, uniform knots $t_1=0, \ldots, t_{10}=90$, $A_0, \ldots, A_5=1000$, -800, 0, 800, -1000, 0. $I=[40,50]$, $x=40$. Here the various individual B-splines (and their even order derivatives) should cancel at x. Note for examples that have symmetry in data like this one, the Full de Boor is always exact, whereas such symmetry is destroyed in the second stages of Boehm and of Lee (i.e., on the faces DAC or DBC).

```
    BOEHM1  :

0  :        -0.181580E-02            -0.408562073062O6E-12
1  :         0.116666E+02             0.11666666666667E+02
2  :         0.596046E-07             0.27755575615629E-16
3  :        -0.433333E-01            -0.43333333333333E-01
4  :         0.000000E+00             0.00000000000000E+00
5  :         0.666666E-03             0.66666666666667E-03

    BOEHM2  :

0  :         0.505066E-02             0.92725827016693E-12
1  :         0.116665E+02             0.11666666666667E+02
2  :         0.178814E-05            -0.83266726846887E-16
3  :        -0.433333E-01            -0.43333333333333E-01
4  :         0.372529E-08             0.86736173798840E-18
5  :         0.666666E-03             0.66666666666667E-03

    LEE  :

0  :         0.000000E+00             0.00000000000000E+00
1  :         0.116667E+02             0.11666666666667E+02
2  :         0.286102E-06             0.66613381477509E-16
3  :        -0.433333E-01            -0.43333333333333E-01
4  :         0.000000E+00             0.00000000000000E+00
5  :         0.666666E-03             0.66666666666667E-03

    FULL DE BOOR  :

0  :         0.000000E+00             0.00000000000000E+00
1  :         0.116667E+02             0.11666666666667E+02
2  :         0.000000E+00             0.00000000000000E+00
3  :        -0.433333E-01            -0.43333333333333E-01
4  :         0.000000E+00             0.00000000000000E+00
5  :         0.666666E-03             0.66666666666667E-03
```

Example 4 (Boehm unsatisfactory near a zero): $m=3$, $t_1, \dots, t_6 = -1000, -700, 1, 10, 700, 1000$, $A_0, \dots, A_3 = -100, 0, 0, 500$. $I=[1, 10]$, $x=4.3$.

```
    BOEHM1  :

0  :        -0.564381E-05            -0.10392013150729E-04
1  :         0.136988E-02             0.13698051019351E-02
2  :         0.174223E-03             0.17422329779359E-03
3  :         0.950526E-04             0.95052685626781E-04

    BOEHM2  :

0  :         0.482537E-04            -0.10392013119457E-04
1  :         0.137001E-02             0.13698051019352E-02
2  :         0.174223E-03             0.17422329779359E-03
3  :         0.950526E-04             0.95052685626781E-04

    LEE  :

0  :        -0.103947E-04            -0.10392013146910E-04
1  :         0.136980E-02             0.13698051019351E-02
2  :         0.174223E-03             0.17422329779359E-03
3  :         0.950526E-04             0.95052685626781E-04
```

```
   FULL DE BOOR :

0 :       -0.103947E-04               -0.10392013146910E-04
1 :        0.136980E-02                0.13698051019351E-02
2 :        0.174223E-03                0.17422329779359E-03
3 :        0.950526E-04                0.95052685626781E-04
```

We must point out that in some other cases Lee is also inaccurate. The following data is taken from [5]. (We regret not having tested this before the publication of [6].).

Example 5 (Inaccuracy in Lee): $m=3$, $t_1, \dots, t_6 = 3, 4, 5, 6, 1000, 2000$, $A_0, \dots, A_3 = 0, 100, 0, 0$. $I=[5,6]$, $x=6$. Note inaccuracy in the second derivative. Here we are calculating at the right end of an almost step function, which peaks slightly to the left of x. With higher degrees, this phenomenon becomes even more pronounced.

```
   BOEHM1 :

0 :        0.996989E+02                0.99698896086860E+02
1 :       -0.100296E+00               -0.10030070028859E+00
2 :        0.915527E-04                0.10090613711000E-03
3 :        0.167169E+02                0.16716968376017E+02

   BOEHM2 :

0 :        0.996989E+02                0.99698896086860E+02
1 :       -0.100301E+00               -0.10030070028859E+00
2 :        0.100906E-03                0.10090613711126E-03
3 :        0.167169E+02                0.16716968376017E+02

   LEE :

0 :        0.996989E+02                0.99698896086860E+02
1 :       -0.100311E+00               -0.10030070028860E+00
2 :        0.991821E-04                0.10090613710823E-03
3 :        0.167169E+02                0.16716968376017E+02

   FULL DE BOOR :

0 :        0.996989E+02                0.99698896086860E+02
1 :       -0.100301E+00               -0.10030070028859E+00
2 :        0.100906E-03                0.10090613711126E-03
3 :        0.167169E+02                0.16716968376017E+02
```

The reason for the relative instability of Lee is clear. The algorithm is based on splitting the function at x (by inserting knots of full multiplicity there). Even if the original *B*-coefficients of F are good, the resulting function may have coefficients that are too close in value as to cause numerical problems during differencing (i.e., the second stage of the algorithm). In Example 5 for instance, after splitting at $t=6$, three of the coefficients are almost equal:

$$0, \quad 100, \quad (994/996)\,100, \quad (994/996)(994/995)\,100.$$

It is also clear why the Full de Boor is the most stable. Since forming convex combinations is a stable operation, the only inaccuracy that could occur is in the differencing operations. If errors occur here, they also occur in Boehm, and one would expect that the function resulting from a split would exhibit similar problems in its coefficients. In the tests, the *B*-coefficients are the inputs rather than results from some previous computations; hence one expects little or no error arising from this first stage.

Finally we give an example of high degree. As one would expect, the effect of roundoff errors propagates up toward higher order derivatives in Lee, while in Boehm it filters down toward the lower order derivatives.

Example 6 (A higher degree case): $m=10$, $t_1, \ldots, t_{20} = 0, 0, 0, .1, .1, .1, 1, 2, 3, 4, 5, 6, 7, 8, 8.1, 8.1, 8.1, 9, 9, 9$. $A_0, \ldots, A_{10} = 10, -1, 20, -2, 30, -3, 40, -4, 50, -5, 60$. $I=[4,5]$, $x=4.5$.

```
    BOEHM1  :

 0  :         0.165169E+02                0.16517487826967E+02
 1  :         0.293772E+00                0.29388093730672E+00
 2  :        -0.202379E-01               -0.20221001262447E-01
 3  :        -0.619270E-02               -0.61905375375837E-02
 4  :         0.551254E-03                0.55148937407166E-03
 5  :         0.243299E-02                0.24330338521549E-02
 6  :        ·0.110526E-02                0.11052727374274E-02
 7  :        -0.145986E-02               -0.14598590878091E-02
 8  :        -0.225491E-02               -0.22549130317233E-02
 9  :         0.188835E-02                0.18883491291061E-02
10  :         0.145791E-01                0.14579104053364E-01

    BOEHM2  :

 0  :         0.165163E+02                0.16517487826967E+02
 1  :         0.294044E+00                0.29388093730676E+00
 2  :        -0.202391E-01               -0.20221001262436E-01
 3  :        -0.619079E-02               -0.61905375375826E-02
 4  :         0.552452E-03                0.55148937407197E-03
 5  :         0.243270E-02                0.24330338521549E-02
 6  :         0.110534E-02                0.11052727374274E-02
 7  :        -0.145987E-02               -0.14598590878091E-02
 8  :        -0.225491E-02               -0.22549130317233E-02
 9  :         0.188835E-02                0.18883491291061E-02
10  :         0.145791E-01                0.14579104053364E-01

    LEE  :

 0  :         0.165174E+02                0.16517487826967E+02
 1  :         0.293884E+00                0.29388093730677E+00
 2  :        -0.202230E-01               -0.20221001262442E-01
 3  :        -0.619570E-02               -0.61905375375716E-02
 4  :         0.571102E-03                0.55148937404793E-03
 5  :         0.238652E-02                0.24330338522052E-02
 6  :         0.120352E-02                0.11052727373251E-02
 7  :        -0.165992E-02               -0.14598590876031E-02
 8  :        -0.185242E-02               -0.22549130321362E-02
 9  :         0.108180E-02                0.18883491299326E-02
10  :         0.161932E-01                0.14579104051710E-01

    FULL DE BOOR  :

 0  :         0.165174E+02                0.16517487826967E+02
 1  :         0.293881E+00                0.29388093730677E+00
 2  :        -0.202211E-01               -0.20221001262438E-01
 3  :        -0.619051E-02               -0.61905375375822E-02
 4  :         0.551461E-03                0.55148937407187E-03
 5  :         0.243303E-02                0.24330338521549E-02
 6  :         0.110527E-02                0.11052727374274E-02
 7  :        -0.145986E-02               -0.14598590878091E-02
 8  :        -0.225491E-02               -0.22549130317233E-02
 9  :         0.188835E-02                0.18883491291061E-02
10  :         0.145791E-01                0.14579104053364E-01
```

4. Remarks

We list a few concluding remarks:

1. On machines with shorter word lengths such as IBM or VAX, computations should be carried out in double precision.
2. When converting a B-spline expansion into piecewise polynomial representation, one should use the Full de Boor algorithm. Also, it might be necessary to do several conversions for the same polynomial piece, when the knot interval is large, corresponding to using Taylor expansions with different centers.

   We may also suggest that if the B-spline coefficients are the original input data, it may even be sensible not to convert, unless speed is a paramount issue.
3. One may devise hybrids of the above algorithms. For example, one can have a "Stratified" de Boor, as represented in Fig. 2. Each "stratum" has s "levels" (except perhaps for the topmost statum). De Boor convex combinations are carried out for the base (with thick lines) of each stratum, and then $(s-1)$ derivatives are obtained as in Lee. This of course includes both Lee $(s=m+1)$ and Full de Boor $(s=1)$. We have coded and tested this satisfactorily. For high degrees, with the conservative choice of $s=2$, this eliminates the problem of Lee at higher derivatives, yet is faster than the Full de Boor. Other combinations between any two of the three algorithms are of course possible.
4. One should also mention that for high dimensions N but moderate number of evaluations, the second algorithm of [3] (see also Cox [4]) may be preferable. One computes the values (and derivatives, see also [5]) of the B-splines at x, then simply takes the dot products with various components of the relevant coefficients (A_i).

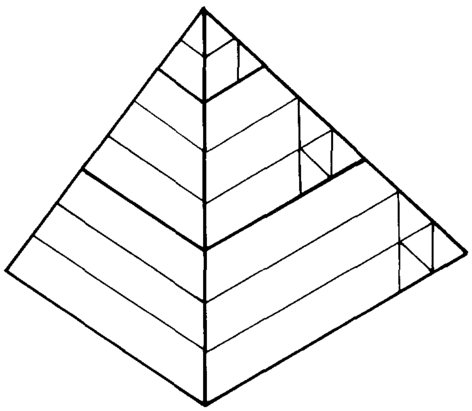

Fig. 2

References

[1] Boehm, W.: Efficient evaluation of splines. Computing *31*, 1 – 7 (1984).
[2] Boehm, W., Farin, G., Kahmann, J.: A survey of curve and surface methods in CAGD. Computer Aided Geometric Design *1*, 1 – 60 (1984).
[3] De Boor, C.: On calculating with *B*-splines. J. Approx. Theory *6*, 50 – 62 (1972).
[4] Cox, M. G.: The numerical evaluation of *B*-splines. J. Inst. Math. Appl. *10*, 134 – 147 (1972).
[5] Butterfield, K. R.: The computation of all derivatives of a *B*-spline basis. J. Inst. Maths Applics *17*, 15 – 25 (1976).
[6] Lee, E. T. Y.: A simplified *B*-spline computation routine. Computing *29*, 365 – 373 (1982).
[7] Germain-Bonne, B., Sablonniere, P.: Comparaison des formes de courbes paramètres et de leurs approximants splines. Publication no. 76 du Laboratoire de Calcul de l'Université de Lille, 1976.

E. T. Y. Lee
Boeing Commercial Airplane Co.
P. O. Box 3707 M/S 6 E-30
Seattle, WA 98124, U.S.A.