

Department of Computer Science The Jniversity of British Columbia Vancouver, British Columbia V6T 1W5

## Abstract

A general purpose collocation code colsys has been written, which is capable of solving mixed order systems of multi-point boundary value ordinary differential equations. The piecewise polynomial solution is given in terms of a $B$-spline basis.

Efficient implementation of algorithms to calculate with B-splines is a necessary condition for the code to be competitive. Here we describe these alqorithms and the special features incorporated to take advantaqe of the specific environment in which they are used.

This paper was written with the financial support of the National Research Council of Canada under grants A 4306 and A 7871.
$\because \quad \rightarrow$

# Evaluation of B-Splines for Solving Systems of Boundary Value Problems 

by<br>U. Ascher ${ }^{\dagger}$ and R. D. Russe11 ${ }^{\dagger \dagger}$

## 1. Introduction

In [1] a general purpose collocation code, COLSYS, for solving mixed order systems of multi-point boundary value ordinary differential equations is described. The piecewise polynomial approximate "global" solution is given in terms of a Bspline basis. This makes the solution process very flexible and stable.

The evaluation of the basis functions is a major cost for COLSYS, and a careful implementation of selected algorithms is needed to make the code competitive. Efficient algorithms for calculating with B-splines are given by deBoor [2], who also efficiently implements these algorithms in a Fortran package [3]. Here we describe the modifications we have made in these algorithms to better suit our needs. Our use of B-splines is somewhat special because:
(i) we are solving a system of differential equations, so many repetitive calculations can be avoided,
(ii) the continuity in the solution at the mesh points (knots) is more restricted here than in $[2,3]$, and
(iii) on many occasions we evaluate the B-splines at points which are placed in. a regular fashion.

[^0]Specifically, suppose we have a set of differential equations on an interval $[a, b]$ of orders $m_{1} \leq m_{2} \leq \ldots \leq m_{d}$. The collocation solution is constructed on a mesh

$$
\begin{equation*}
\pi: a=x_{1}<x_{2}<\ldots<x_{N}<x_{N+1}=b \tag{1.1}
\end{equation*}
$$

with the $n$-th component $v_{n}(x), 1 \leq n \leq d$, belonging to $P_{k+m_{n}, ~} \cap C^{\left(m_{n}-1\right)}[a, b]$, by collocating at the $k$ Gauss-Legendre points in each subinterval $\left[x_{i}, x_{i+1}\right]$ $i=1, \ldots$, N. Here
(1.2) $\quad p_{k+m_{n}}$, $=\left\{w \mid w\right.$ is a polynomial of degree $<k+m_{n}$ on each subinterval $\left.\left(x_{i}, x_{i+1}\right), 1 \leq i \leq N\right\}$.

Let $T=\left\{t_{j}\right\}_{j=1}^{(N+1) k+2 m_{d}}$ be the knot sequence:

$$
t_{j}= \begin{cases}x_{1} & j \leq k+m_{d}  \tag{1.3}\\ x_{i+1} & i k+m_{d}<j \leq(i+1) k+m_{d} \quad i=1, \ldots, N-1 \\ x_{N+1} & N k+m_{d}<j\end{cases}
$$

Let $N_{j, k}$ denote the $j$-th $B$-spline of order $k[2]$. The function $v_{n}$ is given as a linear combination of the $B$-splines

$$
\begin{equation*}
v_{n}(x)=\sum_{j=-k-m_{n}+2}^{N k} \alpha_{j, n^{N}}^{N}, k+m_{n}(x) \tag{1.4}
\end{equation*}
$$

In fact, only $k+m_{n} B$-splines may be nonzero at $x$ : If $x \in\left[t_{i}, t_{i+1}\right)$, then

$$
\begin{equation*}
v_{n}(x)=\sum_{j=-k-m_{n}+1}^{0} \alpha_{i+j, n^{N}}{ }_{i+j, k+m_{n}}(x) \tag{1.5}
\end{equation*}
$$

We consider the efficient implementation of the following procedures for COLSYS: Evaluation of the B-splines at a point (§2), derivatives of the approxi-
mate solution (§3), derivatives of the basis functions (54), highest order derivatives of the solution (§5), and we conclude with a brief summary of the uses of these procedures in the collocation code.
2. Evaluation of the B-splines at a point $x \in\left[t_{i}, t_{i+1}\right)$

This is the evaluation of the $N_{i+j}, k+m_{n}(x)$. deBoor's algorithm reads [2]:

## Algorithm 2A:

Let $\quad N_{i, l}(x)=1$.


The first saving when using B-splines arises because of the recursive way in which they are defined. One call to algorithm 2A produces all the B-splines that we need for all the components of the solution $\underset{\sim}{v} \equiv\left(v_{1}, v_{2}, \ldots, v_{d}\right)$ and their derivatives (as we shall see). The algorithm produces the triangular array


The knot sequence (1.3) has the special structure that each interior mesh point $x_{i}$ is repeated exactly $k$ times. Since $m_{1} \leq \ldots \leq m_{d} \leq k$, this implies that each B-spline has its support on at most 2 subintervals, and some of them have support on one subinterval only. Let $i=I k+m_{d}$. Then $x \in\left[x_{I}, x_{I+1}\right)$, and since $1 \leq j \leq \ell \leq k+m_{d}-1$ in algorithm $2 A$,

$$
\begin{align*}
t_{i+j}-t_{i+j-\ell} & = \begin{cases}h_{I-1}+h_{I} & \text { for } j \leq k \text { and } \ell \geq j+k \\
h_{I} & \text { for } j \leq k \text { and } \ell \leq j+k-1 \\
h_{I}+h_{I+1} & \text { for } k+1 \leq j\end{cases}  \tag{2.1}\\
t_{i+j}-x & = \begin{cases}\rho h_{I} & \text { for } 1 \leq j \leq k \\
\rho h_{I}+h_{I+1} & \text { for } \left.k+1 \leq j \leq k+m_{d}-1 \text { (or } 2 k\right),\end{cases} \tag{2.2}
\end{align*}
$$

where

$$
\begin{equation*}
h_{I}=X_{I+1}-X_{I}, \quad 1 \leq I \leq N \tag{2.3}
\end{equation*}
$$

Therefore, Algorithm 2A becomes
Algorithm 2B: Suppose $\mathrm{x} \in\left[\mathrm{x}_{\mathrm{I}}, \mathrm{X}_{\mathrm{I}+1}\right)$. Set $\mathrm{N}_{\mathrm{i}, 1}(\mathrm{x}) \equiv 1$;

$$
\rho_{1}=\frac{x_{I+1}-x}{x_{I+1}-x_{I-1}} ; \quad \rho_{2}=\frac{x_{I+1}-x}{x_{I+1}-x_{I}} ; \quad \rho_{3}=\frac{x_{I+1}-x}{x_{I+2}-x_{I}} .
$$

(a) for $\ell=1, \ldots, k: N_{i-\ell, \ell+1}(x)=0$
$\begin{array}{ll}\text { Do for } j=1, \ldots, \ell: \\ (*) & N_{i+j-\ell-1, \ell+1}(x)=N_{i+j-\ell-1, \ell+1}(x)+\rho_{2} \\ N_{i+j-\ell, \ell+1}(x)=\left(1-\rho_{2}\right) N_{i+j-\ell, \ell-\ell, \ell}(x)\end{array}$


Our implementation of algorithm 2B was found to be about $50 \%$ faster than the routine BSPLVN in [3] on the IBM 370/168. (This is somewhat machine dependent because a division operation is replaced by a multiplication, and memory references to arrays are saved). Note also that some storage is saved by eliminating the knot sequence.

There is another major saving to be done here. In algorithm $2 \mathrm{~B}, \rho_{2}$ depends only on the relative position of x in its subinterval. But the collocation points have exactly the same relative position in each subinterval. Thus loops (a) and (c) can be evaluated once and stored for each of the $k$ Gaussian points, used in [1], and while assembling the collocation equations, only loops (b) and (d) have to be executed kN times. The same remarks hold for any set of "regular" evaluation points; e.g., the points $x_{I+1 / 6}=x_{I}+1 / 6 h_{I}, x_{I+1 / 3}=x_{I}+1 / 3 h_{I}$ on which intermediate approximations are evaluated for estimating the error (see $\S 2$ of [1]).

In our code, algorithm 2 B is divided into 2 C , which does loops (a) and (c), and 2 D , which does the rest. The total number of B -splines computed is $\frac{1}{2}\left(k+m_{d}\right)\left(k+m_{d}-1\right)$, while that computed in $2 D$ is $m_{d}\left(m_{d}-1\right)$. Thus, if we ignore the one call to 2 C per N "regular" evaluation points, the savings are about $50 \%$, and
more if $k>m_{d}$.
Note that we have not utilized the symmetry of the collocation points (because it enters only in 2C and is not worth the complication in the program) and the nonconvex modification suggested in [4]. Even though we have not yet seen a case in which this nonconvex modification has significantly affected the accuracy in the solution, its improvement in efficiency is small enough that we have decided to play it safe.

## 3. Computation of derivatives of the approximate solution.

The $n$-th component of the approximate solution, $v_{n}(x)$, is given in terms of the coefficients $\alpha_{j, n}$ by (1.4) or (1.5). Its r-th derivative is given by $\left(x \in\left[t_{i}, t_{i+1}\right)\right)$

$$
\begin{equation*}
v_{n}^{(r)}(x)=\left(k+m_{n}-1\right) \ldots\left(k+m_{n}-r\right) \sum_{j=-k-m_{n}+r+1}^{0} \alpha_{i+j, n}^{(r)} N_{i+j, k+m_{n}-r}(x) \tag{3.1}
\end{equation*}
$$

where

$$
\alpha_{i+j, n}^{(r)}= \begin{cases}\alpha_{i+j, n} & r=0  \tag{3.2}\\ \frac{\alpha_{i+j, n}^{(r-1)}-\alpha_{i+j-1, n}^{(r-1)}}{t_{i+j+k+m_{n}-r}^{-t_{i+j}}} & r>0\end{cases}
$$

In [3] a routine is given which prepares a table of divided differences, according to (3.2) with $\alpha_{i+j, n}^{(r)}$ multiplied by $\left(k+m_{n}-1\right) \ldots\left(k+m_{n}-r\right)$. We have written a similar routine which again uses the fact that $\frac{1}{t_{i+j+k+m_{n}}-r^{-t}{ }_{i+j}}$ is known:

$$
t_{i+j+k+m_{n}-r}-t_{i+j}= \begin{cases}x_{2}-x_{1} & r+1 \leq i+j \leq m_{n}  \tag{3.3}\\ x_{I+1}-x_{I} & (I-1) k+m_{n}+1 \leq i+j \leq I k+r \\ x_{I+2}-x_{I} & I k+r+1 \leq i+j \leq I k+m_{n} \\ I=1, \ldots, N\end{cases}
$$

To compute $\underset{\sim}{z}(\underset{\sim}{v}) \equiv\left(v_{1}, v_{1}^{\prime}, \ldots, v_{1}^{\left(m_{1}-1\right)}, v_{2}, \ldots, v_{d}, \ldots, v_{d}^{\left(m_{d}-1\right)}\right)$ we need terivatives up to order $m_{n}-1$. Thus the resulting algorithm is

Algorithm 3A (with $\alpha_{i+j, n}^{(0)} \equiv \alpha_{i+j, n}$ ):


There are many occasions in COLSYS where $\underset{\sim}{z}(\underset{\sim}{v})$ has to be evaluated. The most frequent evaluations are made during the Newton iterations for a nonlinear problem, where the solution from the former iteration is evaluated to determine the coefficients of the new linearized problem. Other evaluations are made for estimating the error and producing the final approximate solution for the user.

There are two alternative ways to evaluate $\underset{\sim}{z}(\underset{\sim}{v})$.
Algorithm 3B
(a) Generate once the divided difference table for $\alpha_{i, n}^{(r)}, i=1, \ldots, k N+m_{n}$, $\mathrm{r}=1, \ldots, \mathrm{~m}_{\mathrm{n}}-1, \mathrm{n}=1, \ldots, \mathrm{~d}$, using algorithm 3A.
(b) For each $x, x \in\left[t_{i}, t_{i+1}\right.$ ) form the nonzero B-splines, up to order $k+m_{d}$, using algorithm 2B.
(c) (For each $x$ ), form $v_{n}^{(r)}(x)=\sum_{j=-k-m_{n}+r+1}^{0}{ }_{i+j, n}^{(r)} N_{i+j, k+m_{n}-r}^{(x)}$ $\mathrm{r}=0, \ldots, \mathrm{~m}_{\mathrm{n}}-1, \mathrm{n}=1, \ldots, \mathrm{~d}$.

## Algorithm 3C

(a) Generate once the piecewise-polynomial coefficients at the knots: for $x=x_{1}, x_{2}, \ldots, x_{N}$ do algorithm 3B, with $r$ in algorithm $3 A$ going up to
(b) For each $x, x \in\left[x_{I}, x_{I+1}\right)$, form $v_{n}^{(r)}(x)=\sum_{j=r}^{k+m_{n}-1} \frac{v_{n}^{(j)}\left(x_{I}\right)}{(j-r)!}\left(x-x_{I}\right)^{j-r}$.

One disadvantage of $3 C$ as compared to $3 B$ is obviously that the amount of storage required for the divided difference table is more than doubled. It also requires more initialization and, in fact, if the number of points $x$ at which $\underset{\sim}{z}(\underset{\sim}{v}(x))$ is required is less than $N$, algorithm $3 B$ must be more effective. However, when the number of points $\mathbf{x}$ is very large, the situation is different. In the collocation example given in [3], it is algorithm 3C which is used for the nonlinear iterations.

We shall compare the relative efficiency of the two algorithms in our setting. For this, consider two cases.
(i) The approximation is to be evaluated at $M_{1}$ points $x$, irregularly distributed in the interval [a,b].
(ii) The approximation is to be evaluated at $M_{1}=M_{2} \times N$ regularly distributed
points $x, M_{2}$ points per subinterval at the same relative positions: $x-x_{I}=\rho\left(x_{I+1}-x_{I}\right)$ where there are only $M_{2}$ different $\rho$ 's. Consider $N$ large, compared to $M_{2}, k$, and $d$.

First, algorithm 3A involves 2 N divisions, $2 \mathrm{~N}+\mathrm{kN}+\mathrm{m}_{\mathrm{n}}-\mathrm{r}$ subtractions and $k N+m_{n}-r$ multiplications for each $n$ and $r$. This amounts to about ( $k+2$ ) $N$ ( $m^{*}-d$ ) multiplications for $3 B$, and about $(k+2) N\left(m^{*}-d\right)+(k+2) N k d$ multiplications for $3 C$, where $r$ goes up to $k+m_{n}-1$. Here, $m^{*}=\sum_{n=1}^{d} m_{n}$.

For step (b) in algorithm 3B we have about $M_{1}\left(k+m_{d}\right)\left(k+m_{d}-1\right)$ operations for case (i), and $2 M_{1} m_{d}\left(m_{d}-1\right)$ operations for case (ii), where the $2 C$ part can be ignored.

Step (c) in algorithm 3B has

$$
M_{1_{n=1}} \sum_{\sum_{r=0}^{d}}^{m_{n}^{-1}}\left(k+m_{n}-r\right)=M_{1} \sum_{n=1}^{d}\left[\left(k+\frac{1}{2}\right) m_{n}+\frac{1}{2} n_{n}^{2}{ }_{n}^{2}\right]=M_{1}\left[\left(k+\frac{1}{2}\right) m *+\frac{1}{2} \hat{M}\right]=M_{1} \bar{M}
$$

multiplications, where $\hat{M}=\sum_{n=1}^{d} m_{n}^{2}$ and $\bar{M}=\left(k+\frac{1}{2}\right) m *+\frac{1}{2} \hat{M}$.

Thus the overall amount of work measured by the approximate number of multiplications for algorithm 3B is

$$
\begin{equation*}
W_{1}=\left(m^{*}-d\right)(k+2) N+M_{1}\left[\left(k+m_{d}\right)\left(k+m_{d}-1\right)+\bar{M}\right] \tag{3.4}
\end{equation*}
$$

in case (i), and

$$
\begin{equation*}
W_{2}=\dot{N}\left\{\left(m^{*}-d\right)(k+2)+M_{2}\left[2\left(m_{d}^{2}-m_{d}\right)+\bar{M}\right]\right\} \tag{3.5}
\end{equation*}
$$

in case (ii).

For step (b) of algorithm 3C the number of multiplications is $M_{1}\left[\vec{M}+2\left(k+m_{d}-2\right)\right]$. Thus the overall amount of work for algorithm $3 C$ is

$$
\begin{align*}
\bar{W}= & N\left\{\left(m^{*}-d\right)(k+2)+k d(k+2)+2\left(m_{d}^{2}-m_{d}\right)+\bar{M}+\frac{d}{2} k(k+1)\right\}+  \tag{3.6}\\
& M_{1}\left[\bar{M}+2\left(k+m_{d}-2\right)\right] .
\end{align*}
$$

We consider now when $\bar{W}$ is cheaper than $W_{1}$ or $W_{2}$.
Case (i). $\bar{W} \leq W_{1}$.

$$
\mathrm{N}\left\{k d(k+2)+2\left(\mathrm{~m}_{\mathrm{d}}^{2}-\mathrm{m}_{\mathrm{d}}\right)+\bar{M}+\frac{\mathrm{d}_{2}}{2} k(k+1)\right\} \leq M_{1}\left[\left(k+\mathrm{m}_{\mathrm{d}}\right)\left(k+\mathrm{m}_{\mathrm{d}}-1\right)-2\left(k+\mathrm{m}_{\mathrm{d}}-2\right)\right]
$$

Since the right hand term is positive, $\bar{W} \leq W_{1}$ only when $M_{1} \geq N \lambda$, where

$$
\begin{equation*}
\lambda=\frac{k d(k+2)+2\left(m_{d}^{2}-m_{d}\right)+\overline{\mathrm{M}}+\frac{\mathrm{d}}{2} \mathrm{k}(\mathrm{k}+1)}{\left(\mathrm{k}+\mathrm{m}_{\mathrm{d}}\right)\left(\mathrm{k}+\mathrm{m}_{\mathrm{d}}-1\right)-2\left(\mathrm{k}+\mathrm{m}_{\mathrm{d}}{ }^{-2)}\right.} . \tag{3.7}
\end{equation*}
$$

Note that $\lambda$ grows with $d$. For $d=1, m=2$ and $k=3, \lambda=\frac{34}{14} \approx 2.5$. For $d=3$, $m_{1}=1, m_{2}=3=m_{3}$ and $k=4, \lambda=\frac{155}{32} \approx 4.8$.

Case (ii). $\overline{\mathrm{W}} \leq \mathrm{W}_{2}$.

$$
k d(k+2)+2\left(m_{d}^{2}-m_{d}\right)+\bar{M}+\frac{d_{2}}{2} k(k+1) \leq M_{2}\left[2\left(m_{d}^{2}-m_{d}\right)-2\left(k+m_{d}-2\right)\right]
$$

For this to happen we need $2\left(\mathrm{~m}_{\mathrm{d}}^{2}-\mathrm{m}_{\mathrm{d}}\right)-2\left(\mathrm{k}+\mathrm{m}_{\mathrm{d}}-2\right)>0$. Let

$$
\begin{equation*}
\mu=\frac{k d(k+2)+2\left(m_{d}^{2}-m_{d}\right)+\bar{M}+\frac{d}{2} k(k+1)}{2\left[m_{d}^{2}-2 m_{d}-k+2\right]} \tag{3.8}
\end{equation*}
$$

Then algorithm 3C is faster only when $\mu>0$ and $M_{2} \geq \mu$. This is never the case for $m_{d} \leq 2!$ For $d=1, m_{d}=3$ and $k=4, \mu=\frac{64}{2}=32$ and $\mu$ grows with $d$. For $d=1, m_{d}=4$ and $k=5, \mu=\frac{104}{10}=10.4$.

Thus, in practice algorithm 3B is always preferable to algorithm 3C in the "regular" case. This is the case in the nonlinear Newton iterations in COLSYS, except for the first iteration where the former mesh is not the same as the current one.

Based on the above considerations, our code never converts to the plecewise polynomial representation 3C. The additional storage and computational and programming overhead are not compensated in general by speed.

## 4. Derivatives of $B-s p l i n e s$.

In order to assemble the collocation equations, an algorithm Is needed to evaluate derivatives of the beasis functions. The desired output is (x $\varepsilon\left[t_{1,}, t_{i+1}\right.$ ) )



Of course, the formula (3.1) for derivatives of a B-spline combination can be used here, with $\alpha_{i+j, n}^{(0)}=\delta_{j \ell}$ for $N_{i+\ell, k+m_{n}}$. The resulting algorithm follows.

## Algorithm 4A:

It is apparent that there are a number of savings that can be made to improve algorithm 4A. First, if $m_{n}=m_{n+1}$, there is no need to repeat the computations for both orders, since they are the same. Hence, in the beginning the program isolates the set of strictly increasing orders and applies the algorithm only to them. The output array is then filled by copying available data where needed. We shall assume henceforth, without restricting generality, that $m_{1}<m_{2}<\ldots<m_{d}$. Second, note that most of the $\alpha_{i+j, m, \ell}^{(r)}$ are zeros, with a very simple zero structure. By moving the loop on $l$ inside the loop on $j$, the loop on $l$ goes only from $j$ to $j+r$. This has been noted and implemented in [3].

Now, again apply our knowledge of $t_{i+j+k+m_{n}}-r_{i+j}$ :

$$
t_{i+j}-t_{i+j-k-m_{n}+r}= \begin{cases}h_{I-1}+h_{I} & j \leq m_{n}-r  \tag{4.1}\\ h_{I} & m_{n}-r+1 \leq j \leq k \\ h_{I}+h_{I+1} & k+1 \leq j \leq k+m_{n}-r\end{cases}
$$

where we shift the index $j$ to run from 1 to $k+m_{n}-r$ and break it into three domalns, so that in each the divided difference involves multiplication by the same constant (similar to algorithm 2B).

The fourth improvement uses the fact that we have a system of differential equations. If the algorithm is performed for $n=d$, then a number of the $\alpha_{i+j, n, \ell}^{(r)}$ may be determined directly by making the observation that

$$
\alpha_{i+j, n, \ell}^{(r)}= \begin{cases}\alpha_{i+j+\left(m_{d}-m_{n}\right), d, \ell}^{(r)} & j=1, \ldots, k-\left(m_{d}-m_{n}\right)-r+1  \tag{4.2}\\ \alpha_{i+j, d, \ell}^{(r)} & j=m_{d}, \ldots, k+m_{n}-r .\end{cases}
$$

A11 of these points are incorporated in algorithm 4B, which is implemented in the code COLSYS.

Algorithm 4B: (Suppose $\alpha_{i+j, n, \ell}^{(0)}=\delta_{j \ell}$ and $D^{r} N_{j, k+m_{n}}(x), r \geq 1$, are initialized to zero ahead of time. Assume $x \in\left[t_{i}, t_{i+1}\right)=\left[x_{I}, x_{I+1}\right)$ and omit the explicit dependence of $N_{j, k+m_{n}}$ on $x$.

1. Set $\rho_{1}=\frac{1}{x_{I+1}-x_{I-1}} ; \quad \rho_{2}=\frac{1}{x_{I+1}-x_{I}} ; \quad \rho_{3}=\frac{1}{x_{I+2}-x_{I}}$.
2. Do for $r=1, \ldots, m_{d}$ :
(each derivative)
3. 

Do for $j=1, \ldots, m_{d}^{-r}$ :
(largest order $\mathrm{n}=\mathrm{d}$ )
(*)

$$
\begin{aligned}
& \alpha_{i+j+r, d, \ell}^{(r)}=\rho_{1} \times\left[\alpha_{i+j+r, d, \ell}^{(r-1)}-\alpha_{i+j+r-1, d, \ell]}^{(r-1)}\right. \\
& D^{r} N_{i+\ell-k-m_{d}}, k+m_{d}=D^{r} N_{i+\ell-k-m_{d}, k+m_{d}}+\alpha_{i+j+r, d}^{(r)}, N_{i+j-\left(k+m_{d}-r\right), k+m_{d}-r}
\end{aligned}
$$

Do for $j=m_{d}^{-r+1}, \ldots, k$ :
(*) with $\rho_{2}$ replacing $\rho_{1}$
Do for $j=k+1, \ldots, k+m{ }_{d}-r$ :
(*) with $\rho_{3}$ replacing $\rho_{1}$
4.

If $\mathrm{d}=1$, go to step 5.
(other components)
Do for $n=d-1, d-2, \ldots, 1$ :
If $r>m_{n}$ then go to step 5 .
4.a

Do for $j=1, \ldots, \min \left(k+m_{n}-r-m_{d}+1, m_{d}-1\right)$ : ( $\alpha^{\prime}$ s computed above)
Do for $l=j, \ldots, j+r$ :
(**) $\mid \quad D^{r} N_{i+\ell-k-m_{n}}, k+m_{n}=D^{r} N_{i+\ell-k-m_{n}}, k+m_{n}+\alpha_{i+j+r+\left(m_{d}-m_{n}\right), d, \ell}^{(r)}$
$\left.\times N_{i+j-\left(k+m_{n}\right.}-r\right), k+m_{n}-r$
Do for $j=m_{d}, \ldots, k+m_{n}-r$ :
$L(* *)$ with 0 replacing $m_{d}-m_{n}$ in the $\alpha$ index.
4.b

```
5. Do for \(\mathrm{n}=1, \ldots\), d :
    \(c=1\)
    Do for \(r=1, \ldots, m_{n}\) :
\(c=c\left(k+m_{n}-r\right)\)
    Do for \(l=1, \ldots, k+m_{n}\) :
    \(D^{r} N_{i+\ell-k-m_{n}}, k+m_{n}=D^{r} N_{i}+\ell-k-m_{n}, k+m_{n} \times C\)
```

                                    (multiply by constant)
    
## 5. Derivatives of highest order

When selecting a new mesh we need the values of the piecewise constant functions $v_{n}^{\left(k+m_{n}-1\right)}(x) \quad n=1, \ldots, d$. These are obtained in a straightforward manner, starting with the values $\alpha_{i+j, n}^{\left(m_{n}-1\right)}, j=-k, \ldots, 0$, previously obtained in algorithm 3A, and using (3.2), with $t_{i+j+k+m_{n}-r}-t_{i+j}=h_{I}$, to obtain the values $\alpha_{i, n}^{\left(k+m_{n}-1\right)}=v_{n}^{\left(k+m_{n}-1\right)}(x)$ for $x^{n} \varepsilon\left[t_{i}, t_{i+1}\right)=\left[x_{I}, x_{I+1}\right)$.
6. Using the B-spline algorithms in COLSYS

The use of the B-spline routines in the collocation code is summarized as follows:

1. Call 2C for each of the Gaussian points.
2. For each Newton iterate, evaluate the former solution and its derivatives using $3 A$ and $3 B$ (the evaluation with $3 A$ is done beforehand and also serves to check convergence of the nonlinear iteration). Calculate B-splines with 2D and derivatives with 4 B .
3. Call 2C once for points at which the error is checked. Evaluate the solution and derivatives for error estimation on the given mesh using 2D, $3 A$ and $3 B$.
4. Evaluate $\mathrm{v}_{\mathrm{n}}{ }^{\left(k+m_{n}-1\right)}, \mathrm{n}=1, \ldots, d$, as described in section 5 to select the new mesh.
5. When the error tolerances are satisfied, evaluate $\underset{\sim}{\mathcal{N}}(\mathrm{x})$ and derivatives at user-specified points using algorithm 3B.

## References.

1. U. Ascher, J. Christiansen and R. D. Russell, A collocation solver for mixed order systems of boundary value problems, Tech. Rep. \#13, Computer Sclence, UBC, Vancouver, Canada (1977).
2. C. de Boor, On calculating with B-splines, J. Approx, Th., 6 (1972), 50-62.
3. C. de Boor, Package for calculating with B-splines, SIAM J. Numer. Anal., 14 (1977), 441-472.
4. R. D. Russell, Efficiencies of B-splines methods for solving differential equations, Proc. 5th Conf. on Numerical Math., Manitoba (1975), 599-617.

[^0]:    $\dagger$ Computer Science Department, University of British Columbia. The work of this author was supported in part by NRC Grant 非A4306.
    †† Mathematics Department, Simon Fraser University. The work of this author was supported in part by NRC Grant \#A7871.

