# A general matrix representation for non-uniform B-spline subdivision with boundary control

G. Casciola [a], L. Romani [a]

[a] *Department of Mathematics, University of Bologna, P.zza di Porta San Donato 5, 40127 Bologna, Italy*

**Abstract**

Boundary conditions are still an open question in the field of approximating subdivision since the problem of determining a general construction of the endpoint rules we need when subdividing a B-spline curve/surface with Bézier end conditions has not been solved yet.

This consideration prompted us to present an efficient algorithm for the conversion between B-spline bases defined over different knot-partitions, which turns out to be extremely useful for computing a general formulation of the subdivision matrix generating an endpoint-interpolating B-spline of arbitrary degree.

*Key words:* Spline-to-spline conversion; Cox-de Boor recurrence; Knot-insertion; Non-uniform B-spline subdivision; Bézier end conditions.

## 1 Introduction

One of the most outstanding problems in approximating subdivision is that the boundaries of the subdivision surfaces are not confined to be on the boundaries defined by the original control mesh. In the literature this problem has been solved either by extending the boundary vertices of the control mesh or by applying different subdivision rules to the boundary edges and vertices. However these strategies do not always guarantee that the boundary curves defined by the boundary edges are the boundary curves of the limiting subdivision surface. This may cause problems when we have to join two subdivision surfaces with intermediate control meshes, since two connected surfaces may have to overlap at any finite step of subdivision.
Aim of this paper is therefore giving a general formulation of a non-uniform spline-based subdivision scheme with Bézier end conditions.

*18 January 2007*

For any scheme of this type, the subdivision matrix describing the refinement process can be derived from the corresponding spline-to-spline (*bs2bs*, for short) conversion matrix. Such a matrix can be constructed by an easy-to-use algorithm based on the Cox-de Boor recursive formula (de Boor, 1972). The same algorithm can be also exploited in knot-insertion processes as an interesting alternative to existing methods. In fact, it is both computationally cheap (the *bs2bs* matrix construction of a degree-$d$ spline requires only $O(d^2)$ floating point operations, like it happens with Mørken's formulation (Mørken, 1989)) of the generalized Oslo algorithm (Lyche et al., 1993)) and conceptually straightforward.

## 2 A recursive matrix formula for converting non-uniform B-splines on a single knot interval

Knot insertion processes are special cases of the more general problem of transforming a spline between B-spline representations on two arbitrary knot vectors. Obviously, the only splines that can be transformed exactly in this way are those that lie in both spaces. The key to the settlement of this problem lies therefore in finding the coefficients of the non-uniform degree-$d$ B-splines providing the representation of a polynomial piece of degree $d$ into the new degree-$d$ B-spline basis. The solution proposed here is based on the Cox-de Boor recursive formula (de Boor, 1972). More precisely, we derive a very general recursively generated coefficient matrix that allows us to represent an arbitrary degree polynomial in a B-spline basis over a new B-spline basis of the same degree.

Let $\mathbf{t}_k = \{t_{k+j-d},\ \cdots,\ t_k,\ t_{k+1},\ \cdots,\ t_{k+j+1}\}$ be an arbitrary knot-partition defining the degree-$d$, $d \geq 1$, non-uniform B-spline basis function $B_{k+j-d,d,\mathbf{t}_k}(x)$ and let $\mathbf{u}_\ell = \{u_{\ell+i-d},\ \cdots,\ u_\ell,\ u_{\ell+1},\ \cdots,\ u_{\ell+i+1}\}$ be the arbitrary knot-partition defining the degree-$d$, $d \geq 1$, non-uniform B-spline basis function $B_{\ell+i-d,d,\mathbf{u}_\ell}(x)$. Then, for any $j = 0, \cdots, d$, let $B_{k+j-d,d,\mathbf{t}_k}(x)$ denote the $d+1$ basis functions overlapping the single-span interval $[t_k, t_{k+1})$, for an arbitrary $k \in \{d+1, \cdots, \sharp\mathbf{t}_k - d - 1\}$ (see Fig.1).

Analogously, let $B_{\ell+i-d,d,\mathbf{u}_\ell}(x)$, $i = 0, \cdots, d$, denote the $d+1$ basis functions overlapping the single-span interval $[u_\ell, u_{\ell+1})$, for an arbitrary $\ell \in \{d+1, \cdots, \sharp\mathbf{u}_\ell - d - 1\}$.

The key idea behind the polynomial transformation described above is to express each $B_{k+j-d,d,\mathbf{t}_k}(x)$, $j = 0, \cdots, d$ as a linear combination of the degree-$d$ B-splines $B_{\ell+i-d,d,\mathbf{u}_\ell}(x)$, $i = 0, \cdots, d$, by introducing for any $j = 0, \cdots, d$ a set of coefficients $\left\{s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d}\right\}_{i=0,\cdots,d}$ such that for any $x \in \mathbb{R}$

$$B_{k+j-d,d,\mathbf{t}_k}(x) = \sum_{i=0}^{d} s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d}\ B_{\ell+i-d,d,\mathbf{u}_\ell}(x) \qquad \forall j = 0, \cdots, d. \tag{1}$$

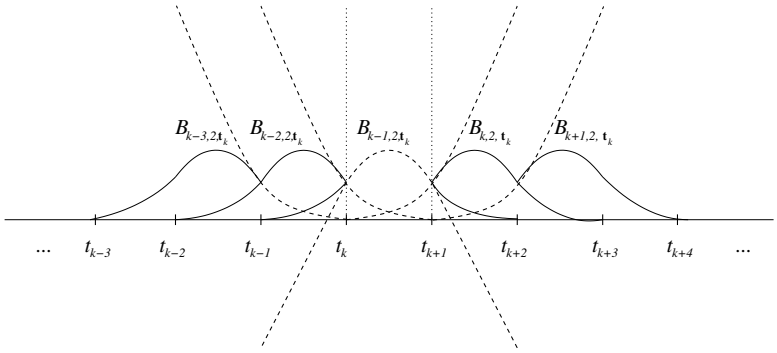


Fig. 1. Non-uniform quadratic B-splines $B_{k+j-2,2,\mathbf{t}_k}(x)$ for $j = -1, 0, 1, 2, 3$.

Then, introducing vectors of basis functions, we can rewrite the transformation in (1) in a convenient matrix form as

$$[\, B_{k-d,d,\mathbf{t}_k}(x) \quad \cdots \quad B_{k,d,\mathbf{t}_k}(x) \,] = [\, B_{\ell-d,d,\mathbf{u}_\ell}(x) \quad \cdots \quad B_{\ell,d,\mathbf{u}_\ell}(x) \,]\, S^{\mathbf{t}_k,\mathbf{u}_\ell,d}. \quad (2)$$

In this way $S^{\mathbf{t}_k,\mathbf{u}_\ell,d} = \left\{ s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d} \right\}_{i,j=0,\cdots,d}$ is the $(d+1) \times (d+1)$ matrix by which we can multiply the column vector of control points of a given degree-$d$ polynomial in the basis $\{B_{k+j-d,d,\mathbf{t}_k}(x)\}_{j=0,\ldots,d}$, to obtain the control points of the same polynomial expressed in the new spline basis $\{B_{\ell+i-d,d,\mathbf{u}_\ell}(x)\}_{i=0,\ldots,d}$. We now focus our attention on the determination of the entries of $S^{\mathbf{t}_k,\mathbf{u}_\ell,d}$ for arbitrarily large $d$ by recurrence over the polynomial degree. Since the non-uniform degree-$d$ B-splines $B_{k+j-d,d,\mathbf{t}_k}(x)$ with $d \geq 1$ turn out to be defined by the Cox-de Boor recursion formula (de Boor, 1972)

$$B_{k+j-d,d,\mathbf{t}_k}(x) = \tfrac{x-t_{k+j-d}}{t_{k+j}-t_{k+j-d}} B_{k+j-d,d-1,\mathbf{t}_k}(x) + \tfrac{t_{k+j+1}-x}{t_{k+j+1}-t_{k+j+1-d}} B_{k+j+1-d,d-1,\mathbf{t}_k}(x)\,, \quad (3)$$

by replacing in (3) the new B-spline representation given in (1) for the $d+1$ B-spline basis functions over the interval $[u_\ell, u_{\ell+1})$, we obtain the relation

$$\sum_{i=0}^{d} s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d} B_{\ell+i-d,d,\mathbf{u}_\ell}(x) = \quad (4)$$

$$\sum_{i=0}^{d-1} \left[ \frac{x - t_{k+j-d}}{t_{k+j} - t_{k+j-d}} s_{i,j-1}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} + \frac{t_{k+j+1} - x}{t_{k+j+1} - t_{k+j+1-d}} s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} \right] B_{\ell+i+1-d,d-1,\mathbf{u}_\ell}(x).$$

Now, by substituting in the left hand member of (4) the Cox-de Boor recurrence relation on $B_{\ell+i-d,d,\mathbf{u}_\ell}(x)$,

$$B_{\ell+i-d,d,\mathbf{u}_\ell}(x) = \tfrac{x-u_{\ell+i-d}}{u_{\ell+i}-u_{\ell+i-d}} B_{\ell+i-d,d-1,\mathbf{u}_\ell}(x) + \tfrac{u_{\ell+i+1}-x}{u_{\ell+i+1}-u_{\ell+i+1-d}} B_{\ell+i+1-d,d-1,\mathbf{u}_\ell}(x)\,, \quad (5)$$

and by taking into account that

$$\sum_{i=0}^{d} s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d}\ \frac{x-u_{\ell+i-d}}{u_{\ell+i}-u_{\ell+i-d}}\ B_{\ell+i-d,d-1,\mathbf{u}_\ell}(x) =$$

$$\sum_{i=0}^{d-1} s_{i+1,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d}\ \frac{x-u_{\ell+i+1-d}}{u_{\ell+i+1}-u_{\ell+i+1-d}}\ B_{\ell+i+1-d,d-1,\mathbf{u}_\ell}(x) \tag{6}$$

and

$$\sum_{i=0}^{d} s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d}\ \frac{u_{\ell+i+1}-x}{u_{\ell+i+1}-u_{\ell+i+1-d}}\ B_{\ell+i+1-d,d-1,\mathbf{u}_\ell}(x) =$$

$$\sum_{i=0}^{d-1} s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d}\ \frac{u_{\ell+i+1}-x}{u_{\ell+i+1}-u_{\ell+i+1-d}}\ B_{\ell+i+1-d,d-1,\mathbf{u}_\ell}(x), \tag{7}$$

we get that (4) can be reduced to

$$\sum_{i=0}^{d-1}\left[s_{i+1,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d}\ \frac{x-u_{\ell+i+1-d}}{u_{\ell+i+1}-u_{\ell+i+1-d}}+s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d}\frac{u_{\ell+i+1}-x}{u_{\ell+i+1}-u_{\ell+i+1-d}}\right]\ B_{\ell+i+1-d,d-1,\mathbf{u}_\ell}(x) =$$

$$\sum_{i=0}^{d-1}\left[\frac{x-t_{k+j-d}}{t_{k+j}-t_{k+j-d}}s_{i,j-1}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1}+\frac{t_{k+j+1}-x}{t_{k+j+1}-t_{k+j+1-d}}s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1}\right]B_{\ell+i+1-d,d-1,\mathbf{u}_\ell}(x). \tag{8}$$

In this way, the following expression on the coefficients necessarily follows:

$$s_{i+1,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d}\ \frac{x-u_{\ell+i+1-d}}{u_{\ell+i+1}-u_{\ell+i+1-d}}+s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d}\frac{u_{\ell+i+1}-x}{u_{\ell+i+1}-u_{\ell+i+1-d}} = \tag{9}$$

$$\frac{x-t_{k+j-d}}{t_{k+j}-t_{k+j-d}}\ s_{i,j-1}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1}+\frac{t_{k+j+1}-x}{t_{k+j+1}-t_{k+j+1-d}}\ s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} \qquad \begin{matrix} i=0,...,d-1,\\ j=0,...,d.\end{matrix}$$

Now evaluating (9) at $x=u_{\ell+i+1-d}$, we get

$$s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d}=\frac{u_{\ell+i+1-d}-t_{k+j-d}}{t_{k+j}-t_{k+j-d}}\ s_{i,j-1}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1}+\frac{t_{k+j+1}-u_{\ell+i+1-d}}{t_{k+j+1}-t_{k+j+1-d}}\ s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} \tag{10}$$

$$i=0,\cdots,d-1\quad j=0,\cdots,d,$$

while evaluating (9) at $x=u_{\ell+i+1}$, it follows that

$$s_{i+1,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d}=\frac{u_{\ell+i+1}-t_{k+j-d}}{t_{k+j}-t_{k+j-d}}\ s_{i,j-1}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1}+\frac{t_{k+j+1}-u_{\ell+i+1}}{t_{k+j+1}-t_{k+j+1-d}}\ s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} \tag{11}$$

$$i=0,\cdots,d-1\quad j=0,\cdots,d.$$

Equations (10) and (11) can be unified by the following formula which, for any $h = 1, ..., d$, allows us to generate the $i-$th row of the polynomial conversion matrix $S^{\mathbf{t}_k,\mathbf{u}_\ell,h}$ ($i \in \{0,\dots,h\}$) starting from the initial value $s_{0,0}^{\mathbf{t}_k,\mathbf{u}_\ell,0} = 1$, in the following way:

$$s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,h} = \frac{\overline{u} - t_{k+j-h}}{t_{k+j} - t_{k+j-n}} s_{i,j-1}^{\mathbf{t}_k,\mathbf{u}_\ell,h-1} + \frac{t_{k+j+1} - \overline{u}}{t_{k+j+1} - t_{k+j+1-n}} s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,h-1} \tag{12}$$

where

$$\overline{u} = \begin{cases} u_{\ell+i+1-h} & \text{if} \quad i \leq h-1 \\ u_{\ell+h} & \text{if} \quad i \geq h \end{cases}$$

for any $j = 0, \dots, h$.

Directly from equation (12) we can work out the following computational scheme

$$\begin{array}{ccccccccc} s_{0,0}^{\mathbf{t}_k,\mathbf{u}_\ell,0} & \rightarrow & s_{0,0:1}^{\mathbf{t}_k,\mathbf{u}_\ell,1} & \rightarrow & s_{0,0:2}^{\mathbf{t}_k,\mathbf{u}_\ell,2} & \rightarrow & \cdots & \rightarrow & s_{0,0:d}^{\mathbf{t}_k,\mathbf{u}_\ell,d} \\ & \searrow & & & & & & & \\ & & s_{1,0:1}^{\mathbf{t}_k,\mathbf{u}_\ell,1} & \rightarrow & s_{1,0:2}^{\mathbf{t}_k,\mathbf{u}_\ell,2} & \rightarrow & \cdots & \rightarrow & s_{1,0:d}^{\mathbf{t}_k,\mathbf{u}_\ell,d} \\ & & & \searrow & & & & & \\ & & & & \ddots & & \cdots & \rightarrow & \cdots \\ & & & & & \searrow & & & \\ & & & & & & s_{d-1,0:d-1}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} & \rightarrow & s_{d-1,0:d}^{\mathbf{t}_k,\mathbf{u}_\ell,d} \\ & & & & & & & \searrow & \\ & & & & & & & & s_{d,0:d}^{\mathbf{t}_k,\mathbf{u}_\ell,d} \end{array} \tag{13}$$

where the symbol $s_{i,0:h}^{\mathbf{t}_k,\mathbf{u}_\ell,h}$ stand for the $i-$th row of $S^{\mathbf{t}_k,\mathbf{u}_\ell,h}$, the arrow $\rightarrow$ indicates the application of formula (10) and $\searrow$ of (11). Such a scheme allows us to compute the polynomial conversion matrix $S^{\mathbf{t}_k,\mathbf{u}_\ell,d}$ and all the sequence of matrices $S^{\mathbf{t}_k,\mathbf{u}_\ell,h}$ for any $h = 1, \dots, d$.
Alternative arrangements of such an algorithm can be derived by selecting different paths in (13), that is by considering together equations (10) and (11) in different manners. For example, we observe that by solving the system (10)-(11) subtracting the latter from the first, the column oriented procedure

$$s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d} = s_{i-1,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d} + (u_{\ell+i+1} - u_{\ell+i+1-d}) \left[ \frac{s_{i-1,j-1}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1}}{t_{k+j} - t_{k+j-d}} - \frac{s_{i-1,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1}}{t_{k+j+1} - t_{k+j+1-d}} \right]$$

$$i = 1, \cdots, d \quad j = 0, \cdots d \tag{14}$$

where the starting values $s_{0,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d}\ \forall j=0,...,d$ are computed through (10) with $i=0$, can be worked out.
Common to the above procedures is that the computation of all the entries in the $S^{\mathbf{t}_k,\mathbf{u}_\ell,d}$ matrix requires $O(d^3)$ floating point operations. In order to get a formula competitive with the cheapest knot-insertion algorithm known in the literature (Barry and Goldman, 1993), we have therefore to work out a recurrence that allows us to fill the $S^{\mathbf{t}_k,\mathbf{u}_\ell,d}$ matrix with only $O(d^2)$ floating point operations. To this aim, we solve the system given by equations (10) and (11)

$$\begin{cases} s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d} = \frac{u_{\ell+i+1-d}-t_{k+j-d}}{t_{k+j}-t_{k+j-d}} s_{i,j-1}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} + \frac{t_{k+j+1}-u_{\ell+i+1-d}}{t_{k+j+1}-t_{k+j+1-d}} s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} & i=0,\cdots,d-1 \\ s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d} = \frac{u_{\ell+i}-t_{k+j-d}}{t_{k+j}-t_{k+j-d}} s_{i-1,j-1}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} + \frac{t_{k+j+1}-u_{\ell+i}}{t_{k+j+1}-t_{k+j+1-d}} s_{i-1,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} & i=1,\cdots,d \end{cases} \tag{15}$$

in the following way

$$\frac{u_{\ell+i-(d-1)}-t_{k+j-(d-1)-1}}{t_{k+j}-t_{k+j-(d-1)-1}} s_{i,j-1}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} + \frac{t_{k+j+1}-u_{\ell+i-(d-1)}}{t_{k+j+1}-t_{k+j-(d-1)}} s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} =$$

$$\frac{u_{\ell+i}-t_{k+j-(d-1)-1}}{t_{k+j}-t_{k+j-(d-1)-1}} s_{i-1,j-1}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} + \frac{t_{k+j+1}-u_{\ell+i}}{t_{k+j+1}-t_{k+j-(d-1)}} s_{i-1,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d-1} \qquad i=1,\cdots,d-1$$

thus obtaining

$$s_{i,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d} = \frac{1}{t_{k+j+1}-u_{\ell+i-d}} \left\{ (t_{k+j+1}-u_{\ell+i})\ s_{i-1,j}^{\mathbf{t}_k,\mathbf{u}_\ell,d} + \frac{t_{k+j+1}-t_{k+j-d}}{t_{k+j}-t_{k+j-d-1}} * \right. \tag{16}$$

$$\left. * \left[ (u_{\ell+i}-t_{k+j-d-1})\ s_{i-1,j-1}^{\mathbf{t}_k,\mathbf{u}_\ell,d} - (u_{\ell+i-d}-t_{k+j-d-1})\ s_{i,j-1}^{\mathbf{t}_k,\mathbf{u}_\ell,d} \right] \right\} \qquad \begin{matrix} i=1,\cdots,d \\ j=0,\cdots,d \end{matrix}$$

where the necessary starting values are computed through (10) with $i=0$. This time, directly from equation (16) we can work out the following compu-

tational scheme

$$
\begin{array}{ccccccc}
s_{0,0}^{\mathbf{t}_k,\mathbf{u}_\ell,0} & \rightarrow & s_{0,0:1}^{\mathbf{t}_k,\mathbf{u}_\ell,1} & \rightarrow & s_{0,0:2}^{\mathbf{t}_k,\mathbf{u}_\ell,2} & \rightarrow \cdots \rightarrow & s_{0,0:d}^{\mathbf{t}_k,\mathbf{u}_\ell,d} \\
 & & \downarrow & & \downarrow & & \downarrow \\
 & & s_{1,0:1}^{\mathbf{t}_k,\mathbf{u}_\ell,1} & & s_{1,0:2}^{\mathbf{t}_k,\mathbf{u}_\ell,2} & \cdots & s_{1,0:d}^{\mathbf{t}_k,\mathbf{u}_\ell,d} \\
 & & & & \downarrow & & \downarrow \\
 & & & & s_{2,0:2}^{\mathbf{t}_k,\mathbf{u}_\ell,2} & \cdots & s_{2,0:d}^{\mathbf{t}_k,\mathbf{u}_\ell,d} \\
 & & & & & & \downarrow \\
 & & & & & \ddots & \vdots \\
 & & & & & & \downarrow \\
 & & & & & & s_{d,0:d}^{\mathbf{t}_k,\mathbf{u}_\ell,d}
\end{array}
\tag{17}
$$

where the symbol $s_{i,0:d}^{\mathbf{t}_k,\mathbf{u}_\ell,d}$ stand for the $i-$th row of $S^{\mathbf{t}_k,\mathbf{u}_\ell,d}$, the arrow $\rightarrow$ indicates the application of formula (10) and $\downarrow$ of (16). Thus, through this last procedure one can fill the conversion matrix row by row (top to bottom and left to right), with only $O(d^2)$ floating point operations.

### *2.1 MATLAB implementation*

To efficiently generate the polynomial conversion matrix $S^{\mathbf{t}_k,\mathbf{u}_\ell,d}$ entirely, we have implemented the MATLAB function `bs2bs` which takes as input the degree $d$, the knot-partitions $\mathbf{t}_k$ and $\mathbf{u}_\ell$ with $\sharp\mathbf{t}_k$ and $\sharp\mathbf{u}_\ell$ knots respectively, the indexes $k \in [d+1, \sharp\mathbf{t}_k - d - 1]$ and $\ell \in [d+1, \sharp\mathbf{u}_\ell - d - 1]$ identifying the non-trivial intervals $[t_k, t_{k+1})$ and $[u_\ell, u_{\ell+1})$ such that $[t_k, t_{k+1}) \cap [u_\ell, u_{\ell+1}) \neq \emptyset$.

```
function S=bs2bs(d,t,u,k,l)
S=zeros(d+1);
S(1,:)=bs2bs_first_row(d,t,u,k,l);
for ir=1:d
    S(ir+1,:)=bs2bs_i_row(d,t,u,k,l,ir,S(ir,:));
end
```

In such a function, the first row of the polynomial conversion matrix $S^{\mathbf{t}_k,\mathbf{u}_\ell,d}$ is generated by the module `bs2bs_first_row` (which implements equation (10) with $i = 0$), while the $i-$th row, $i \in \{1, ..., d\}$, is computed by `bs2bs_i_row` (which implements equation (16) for $i = 1, ..., d$).
Their codes are included below.

```
function S=bs2bs_first_row(d,t,u,k,l)
S=eye(1,d+1);
for h=1:d
    beta_2=0.0;
    uu=u(l+1-h);
    for j=h:-1:1
        d1=uu-t(k+j-h);
        d2=t(k+j)-uu;
        beta_1=S(j)/(d2+d1);
        S(j+1)=d1*beta_1+beta_2;
        beta_2=d2*beta_1;
    end
    S(1)=beta_2;
end
```

```
function Si=bs2bs_i_row(d,t,u,k,l,ir,S)
Si(1)=S(1)*(t(k+1)-u(l+ir))/(t(k+1)-u(l+ir-d));
for j=1:d
    den=t(k+j+1)-u(l+ir-d);
    fact=(t(k+j+1)-t(k+j-d))/(t(k+j)-t(k+j-d-1));
    Si(j+1)=(fact*(S(j)*(u(l+ir)-t(k+j-d-1))-Si(j)*
        *(u(l+ir-d)-t(k+j-d-1)))+S(j+1)*(t(k+j+1)-u(l+ir)))/den;
end
```

*An application example*

This kind of polynomial conversion matrix turns out to be extremely useful when we have to resize a circle arc. For example, by simply typing the command $S$=`bs2bs`(2,$t$,$u$,3,3), where $t = [0\ 0\ 0\ 1\ 1\ 1]$ and $u = [-1\ -1\ -1\ 2\ 2\ 2]$, we can compute the 3×3 matrix

$$S = \begin{bmatrix} 4 & -4 & 1 \\ -2 & 5 & -2 \\ 1 & -4 & 4 \end{bmatrix}$$

that allows us to expand the quadratic arc defined over the interval [0,1] to the interval [-1,2] (see Fig.2).

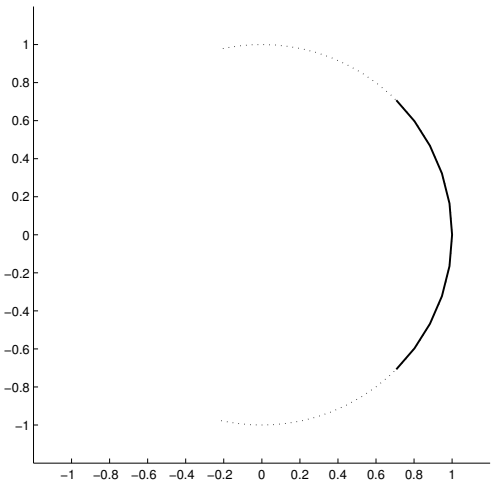


Fig. 2. Example of circle arc resizing.

## 3 A general formulation of the spline-to-spline conversion matrix on an arbitrary sequence of knot intervals

To compute the piecewise polynomial representation of a complete spline curve defined on the knot-partition $\mathbf{t} = \bigcup_k \mathbf{t}_k$, over the new knot-partition $\mathbf{u} = \bigcup_\ell \mathbf{u}_\ell$, we have to build the $(\sharp\mathbf{u} - d - 1) \times (\sharp\mathbf{t} - d - 1)$ rectangular matrix $S^{\mathbf{t},\mathbf{u},d}$ by composing all the polynomial conversion matrices $S^{\mathbf{t}_k,\mathbf{u}_\ell,d}$ in the following way:

$$S^{\mathbf{t},\mathbf{u},d}_{\ell-d:\ell,k-d:k} = S^{\mathbf{t}_k,\mathbf{u}_\ell,d}. \tag{18}$$

The MATLAB function `bs2bs_global` has been devised to solve this problem: it takes as input the degree $d$, the knot-partitions $\mathbf{t}$ and $\mathbf{u}$ with $\sharp\mathbf{t}$ and $\sharp\mathbf{u}$ knots respectively, to compute the whole spline conversion matrix $S^{\mathbf{t},\mathbf{u},d}$.

```
function S=bs2bs_global(d,t,u)
nt=length(t);
nu=length(u);
S=zeros(nu-d-1,nt-d-1);
[t_mult,t_single,nt_s]=knot_mult(d,t);
[u_mult,u_single,nu_s]=knot_mult(d,u);
st=d+1;
su=d+1;
row=1;
col=1;
```

```
Sl=bs2bs(d,t,u,st,su);
S(row:d+row,col:d+col)=Sl;
t_single(nt+1)=t(nt-d);
i=1;
for j=1:nu_s
    if (u_single(j) == t_single(i))
        st=st+t_mult(i);
        col=col+t_mult(i);
        i=i+1;
    end
    su=su+u_mult(j);
    row=row+u_mult(j);
    Sl=bs2bs(d,t,u,st,su);
    S(row:d+row,col:d+col)=Sl;
end
```

In the procedure `bs2bs_global`, an auxiliary function `knot_mult` is called, which, given an extended knot-partition, returns the break points with their multiplicities (`t_single` and `t_mult`, respectively) as well as the cardinality of `t_single` (`nt_s`). We include this straightforward code for completeness.

```
function [t_mult,t_single,nt_s]=knot_mult(d,t)
nt=length(t);
nt_s=0;
m=1;
for i=d+2:nt-d-1
    if (t(i) < t(i+1))
        nt_s=nt_s+1;
        t_mult(nt_s)=m;
        t_single(nt_s)=t(i);
        m=1;
    else
        m=m+1;
    end
end
t_single(nt_s+1)=t(nt-d);
t_mult(nt_s+1)=0;
```

### *3.1 The subdivision matrix of endpoint-interpolating B-splines*

The subdivision matrix of a non-uniform degree-$d$ B-spline interpolating the endpoints can thus be generated through the function `bs2bs_global`($d$,$t$,$u$) where $t$ and $u$ are the knot partitions associated with the coarse and the refined B-spline representations, respectively, characterized by $d + 1$ equal

starting values due to the Bézier end conditions.

*Example 1.* Subdivision rules of a binary cubic B-spline subdivision scheme with Bézier end conditions.

By the command $S$=`bs2bs_global`(3,$t$,$u$), where $t$ = [0 0 0 0 2 4 6 8 10 12] and $u$ = [0 0 0 0 1 2 3 4 5 6 7 8 9], we can compute the 9×6 matrix

$$S = \frac{1}{16}\begin{bmatrix} 16 & 0 & 0 & 0 & 0 & 0 \\ 8 & 8 & 0 & 0 & 0 & 0 \\ 0 & 12 & 4 & 0 & 0 & 0 \\ 0 & 3 & 11 & 2 & 0 & 0 \\ 0 & 0 & 8 & 8 & 0 & 0 \\ 0 & 0 & 2 & 12 & 2 & 0 \\ 0 & 0 & 0 & 8 & 8 & 0 \\ 0 & 0 & 0 & 2 & 12 & 2 \\ 0 & 0 & 0 & 0 & 8 & 8 \end{bmatrix}$$

that allows us to define the binary refinement rules we need when subdividing a cubic B-spline with Bézier end conditions.

*Example 2.* Subdivision rules of a binary quartic B-spline subdivision scheme with Bézier end conditions.

By the command $S$=`bs2bs_global`(4,$t$,$u$), where $t$ = [0 0 0 0 0 2 4 6 8 10 12 14] and $u$ = [0 0 0 0 0 1 2 3 4 5 6 7 8 9 10], we can compute the 10×7 matrix

$$S = \frac{1}{48}\begin{bmatrix} 48 & 0 & 0 & 0 & 0 & 0 & 0 \\ 24 & 24 & 0 & 0 & 0 & 0 & 0 \\ 0 & 36 & 12 & 0 & 0 & 0 & 0 \\ 0 & 9 & 33 & 6 & 0 & 0 & 0 \\ 0 & 0 & 20 & 25 & 3 & 0 & 0 \\ 0 & 0 & 4 & 29 & 15 & 0 & 0 \\ 0 & 0 & 0 & 15 & 30 & 3 & 0 \\ 0 & 0 & 0 & 3 & 30 & 15 & 0 \\ 0 & 0 & 0 & 0 & 15 & 30 & 3 \\ 0 & 0 & 0 & 0 & 3 & 30 & 15 \end{bmatrix}$$

that allows us to define the binary refinement rules we need when subdividing a quartic B-spline with Bézier end conditions.

*Example 3.* Subdivision rules of a ternary cubic B-spline subdivision scheme with Bézier end conditions.

By the command $S$=`bs2bs_global`(3,$t$,$u$), where $t$ = [0 0 0 0 3 6 9 12 15 18] and $u$ = [0 0 0 0 1 2 3 4 5 6 7 8 9 10 11 12], we can compute the 12×6 matrix

$$S = \frac{1}{54}\begin{bmatrix} 54 & 0 & 0 & 0 & 0 & 0 \\ 36 & 18 & 0 & 0 & 0 & 0 \\ 12 & 36 & 6 & 0 & 0 & 0 \\ 0 & 30 & 22 & 2 & 0 & 0 \\ 0 & 12 & 34 & 8 & 0 & 0 \\ 0 & 3 & 31 & 20 & 0 & 0 \\ 0 & 0 & 20 & 32 & 2 & 0 \\ 0 & 0 & 8 & 38 & 8 & 0 \\ 0 & 0 & 2 & 32 & 20 & 0 \\ 0 & 0 & 0 & 20 & 32 & 2 \\ 0 & 0 & 0 & 8 & 38 & 8 \\ 0 & 0 & 0 & 2 & 32 & 20 \end{bmatrix}$$

that allows us to define the ternary refinement rules we need when subdividing a cubic B-spline with Bézier end conditions.

## References


Barry, P.J., Goldman, R.N., 1993. Knot insertion algorithms. In: Goldman, R.N., Lyche, T. (Eds.), Knot-insertion and deletion algorithms for B-spline curves and surfaces. SIAM, Philadelphia, 89-133.

de Boor, C., 1972. On calculating with B-splines. J. Approximation Theory 6, 50-62.

Lyche, T., Mørken, K., Strøm, K., 1993. Conversion between B-spline bases using the generalized Oslo algorithm. In: Goldman, R., N., Lyche, T. (Eds.), Knot-insertion and deletion algorithms for B-spline curves and surfaces. SIAM, Philadelphia, 135-153.

Mørken, K., 1989. Contribution to the theory and applications of splines. Dr. Scient. Thesis, Institutt for Informatikk, University of Oslo, Oslo, Norway.