Cubic B-splines Using PSTricks

Michael Sharpe

msharpe@ucsd.edu

A cubic, uniform B-spline curve with control points $B_0 \dots B_n$ ($n \geq 2$) is a curve parametrized by the interval $[0, n]$, which is, except in degenerate cases, C^2-continuous (that is, has continuous curvature) and is on each interval $[k-1, k]$ ($0 < k \leq n$ an integer) given by a cubic Bézier curve whose control points are derived from the (B_k). These curves are discussed in any reasonably modern text on Numerical Analysis. One easily accessible source is the UCLA lecture notes of Kirby Baker:

`http://www.math.ucla.edu/~baker/149.1.02w/handouts/dd_splines.pdf`

I'll focus on two special cases: (i) relaxed, uniform B-splines; (ii) periodic, uniform B-splines. 'Uniform' refers to the condition mentioned in the first paragraph: each Bézier sub-curve is parametrized by an interval of length 1. 'Relaxed' means that the curvature vanishes at the endpoints $t = 0, t = n$. 'Periodic' means in effect that the B_i repeat periodically, and the curve generated is a closed curve.

1. Quick summary of the macros

`\psbspline(1,1)(3,0)(5,2)(4,5)`: draws the relaxed, uniform B-spline interpolating the specified points.

`\psBspline(1,1)(3,0)(5,2)(4,5)`: draws the relaxed, uniform B-spline with specified control points.

`\psBspline{B}(1,1)(3,0)(5,2)(4,5)`: draws the relaxed, uniform B-spline with specified control points, using `B` as basename for the constructed points.

`\psBsplineE(1,1)(3,0)(5,2)(4,5)`: has the same effect as the command `\psBspline`(1,1)(3,0)(5,2)(4,5) except that it omits the first and last segments.

`\psBsplineC(1,1)(3,0)(5,2)(4,5)`: extends the specified points periodically, drawing a closed curve with the specified points as control points.

`\psBsplineNodes{B}{4}`: draws the relaxed, uniform B-spline with control points `B0`..`B4`.

`\psBsplineNodesE{B}{4}`: is the same as `\psBsplineNodes`{B}{4} except that it omits the first and last segments.

\psBsplineNodesC{B}{4}: extends the node sequence periodically, drawing a closed curve with them as control points.

\beztobsp(1,2)(-3,-4)(5,6)(-7,-8){B}: creates nodes B0..B3 for which the curve \psBsplineNodesE{B}{3} is identical to the Bézier curve determined by the specified points. (It does not draw the curve.)

\bspcurvepoints{B}{5}{P}: creates PostScript arrays to describe a sequence of points along the curve that would be the result of the command \psBsplineNodes{B}{5}, naming those arrays P.X, P.Y (for position), PNormal.X, PNormal.Y, PDelta.X and PDelta.Y. Must be preceded by a \psBsplineNodes command. (Nothing is drawn.)

\bspcurvepointsE{B}{5}{P}: does the same as \bspcurvepoints, but omits the first and last segments. Must be preceded by a \psBsplineNodes[E] command.(Nothing is drawn.)

\bspNode{P}{5}{1.3}{Q}: requires that you first run \bspcurvepoints[E] to create PostScript arrays with basename P. It then sets a node Q at position $t = 1.3$ on the curve. (Nothing is drawn.)

\bspFnNode{P}{5}{2.3}{Q}: requires that you first run \bspcurvepoints[E] to create PostScript arrays with basename P. It then sets a node Q at position $x = 1.3$ on the curve. (Nothing is drawn.) The result is meaningful only for a B-spline curve that is the graph of a function of x and where $x_0 < x_1 < \cdots$.

\psBsplineInterp{S}{4}: will construct a sequence SB0..SB4 for which the associated B-spline curve interpolates S0..S4. (Nothing is drawn—you have to then issue the command \psBsplineNodes{SB}{4}.)

\psBsplineInterpC{S}{4}: will construct a sequence SB0..SB5 for which the associated closed B-spline curve interpolates S0..S4. (Nothing is drawn — you have then to issue the command \psBsplineNodesC{SB}{5}.)

\thickBspline{B}{5}{12pt}{<graphic to clip>}: defines a clipping path 12pt wide around the B-spline curve with control points B0..B5, then draws the <graphic> clipped to that path.

\bspcurvenodes{P}{Q}: creates a node sequence Q0 Q1,... from the position data in the arrays P.X, P.Y created a \bspcurvepoints macro.

Details and examples are provided below.

2. Relaxed, Open B-spline

The algorithm to generate such a curve from a sequence of control points $B_0, \cdots, B_n$ is as follows:

- The curve starts at B_0 and ends at B_n. (Important: $n \geq 2$.)

- Divide each line segment $B_{k-1}B_k$ into equal thirds, with subdivision points labeled R_{k-1}, L_k respectively, so that B_k has L_k as its immediate neighbor to the left, and R_k as its immediate neighbor to the right.
- For $0 < k < n$, divide the line segment L_kR_k in half, letting S_k denote the midpoint. In effect, for $0 < k < n$, $S_k = (B_{k-1} + 4B_k + B_{k+1})/6$.
- Let $S_0 = B_0$ and $S_n = B_n$.
- For $0 < k \leq n$, construct the cubic Bézier curve with control points S_{k-1}, R_{k-1}, L_k, S_k, parametrized by $k - 1 \leq t \leq k$.

The `pst-Bspline` package implements this algorithm as `\psBspline`, whose simplest form is, for example

```
\psBspline(.5,.5)(2,0)(5,2)(6,4)(4,5)(2,4)
```

The coordinates are the B-spline control points. Aside from the usual keywords, like `linestyle`, `linecolor` and `arrows`, there is a Boolean keyword `showframe`. The effect of `showframe=true` is to show the intermediate points and lines in the algorithm described above.

There is another optional argument that can be applied if you wish to be able to refer to any of the points constructed in the algorithm. By example,

```
\psBspline{B}(.5,.5)(2,0)(5,2)(6,4)(4,5)(2,4)
```

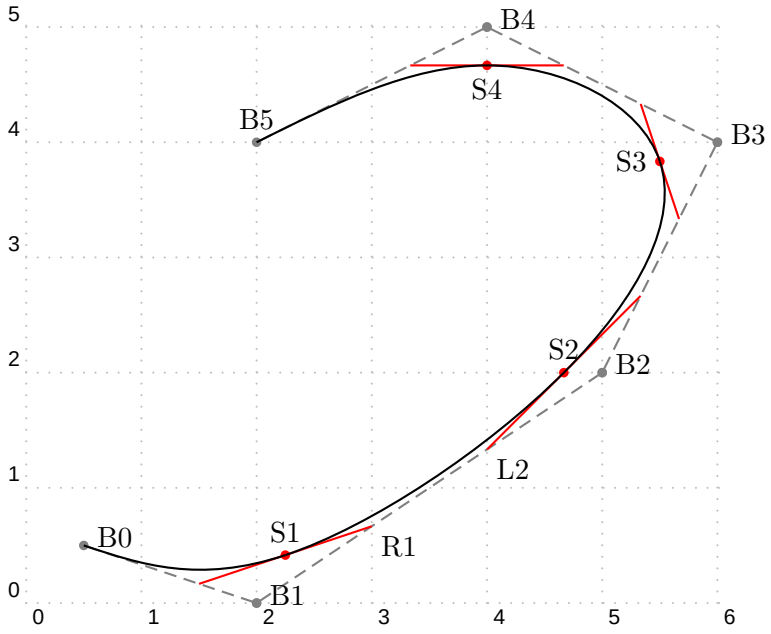
sets the root of the naming scheme to `B`, the effect of which is that the B-spline control points will be nodes of type `\pnode` with names `B0`, `B1` and so on, the other points being similarly named `BL0`, `BL1`, ... , `BR0`, `BR1`, ... , `BS0`, `BS1`, ... For example, to draw a line between `BL1` and `BS4`, just use `\ncline(BL1)(BS4)`.

The algorithm depends for the most part on the flexibility of nodes, and above all the `\multido` macro, which allows one to construct with relative ease items that look and feel like arrays. Use of `\SpecialCoor` is essential.

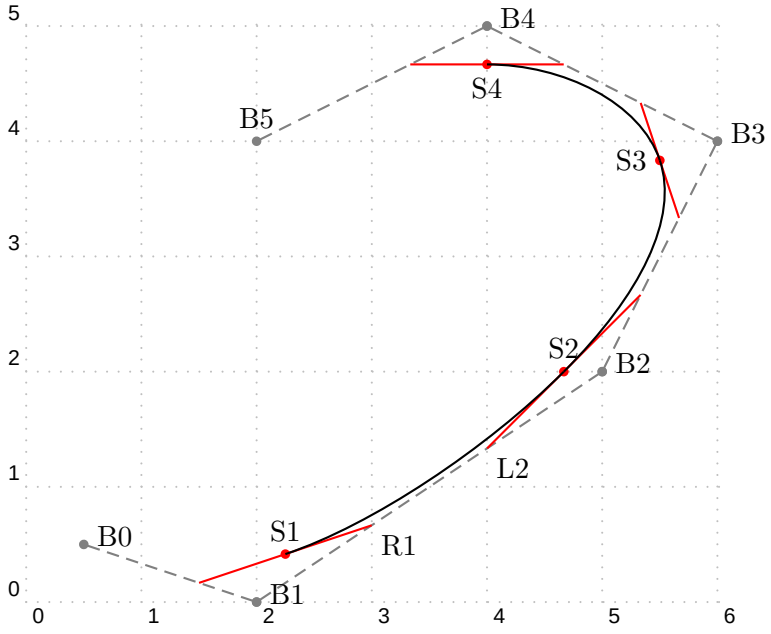
There is a closely related macro `\psBsplineE` which removes the first and last Bézier segments, much as `\psecurve` acts in relation to `\pscurve`, allowing one to draw B-splines with non-zero curvature at the endpoints.

```
\documentclass{article}
\usepackage{pstricks}
\usepackage{multido,pst-node,pst-bspline}
\pagestyle{empty}
\begin{document}
\SpecialCoor % essential for pst-bspline package
\psset{unit=.6in}
```

```
\begin{pspicture}[showgrid=true](-.5,-.5)(6,5)
\psBspline[showframe=true]{B}(.5,.5)(2,0)(5,2)(6,4)(4,5)(2,4)
\multido{\i=0+1}{5}{\uput[20](B\i){B\i}}
\uput[90](B5){B5}\uput[90](BS1){S1}\uput[90](BS2){S2}
\uput[180](BS3){S3}\uput[270](BS4){S4}\uput[-45](BR1){R1}
\uput[-45](BL2){L2}
\end{pspicture}
\end{document}
```



```
\documentclass{article}
\usepackage{pstricks}
\usepackage{multido,pst-node,pst-bspline}
\pagestyle{empty}
\begin{document}
\SpecialCoor % essential for pst-bspline package
\psset{unit=.6in}
\begin{pspicture}[showgrid=true](-.5,-.5)(6,5)
\psBsplineE[showframe=true]{B}(.5,.5)(2,0)(5,2)(6,4)(4,5)(2,4)
\multido{\i=0+1}{5}{\uput[20](B\i){B\i}}
\uput[90](B5){B5}
\uput[90](BS1){S1}
\uput[90](BS2){S2}
\uput[180](BS3){S3}
\uput[270](BS4){S4}
\uput[-45](BR1){R1}
\uput[-45](BL2){L2}
\end{pspicture}
\end{document}
```



2.1. **Bézier curves as B-spline curves.** Consider a Bézier curve $\mathscr{C}$ with control points S_1, C_1, C_2, S_2. To identify $\mathscr{C}$ as a B-spline curve of the type discussed above, consider the problem of finding the B-spline control points B_0, B_1, B_2, B_3 for which `\psBsplineE` yields $\mathscr{C}$. This is a simple problem in linear algebra whose solution is:

```
B0=6S1-7C1+2C2
B1=2C1-C2
B2=-C1+2C2
B3=6S2+2C1-7C2
```

In other words, any cubic Bézier curve may be considered to be the output of `\psBsplineE` for the B_i described above. In this way, all macros described for B-spline curves may be applied to an arbitrary Bézier curve as a special case. The macro

```
\beztobsp(S1)(C1)(C2)(S2){B}
```

results in defining $B_0,\ldots,B_3$ exactly as above.

3. Periodic B-spline

The result here is a closed curve. The algorithm is essentially the same as in the preceding case, except:

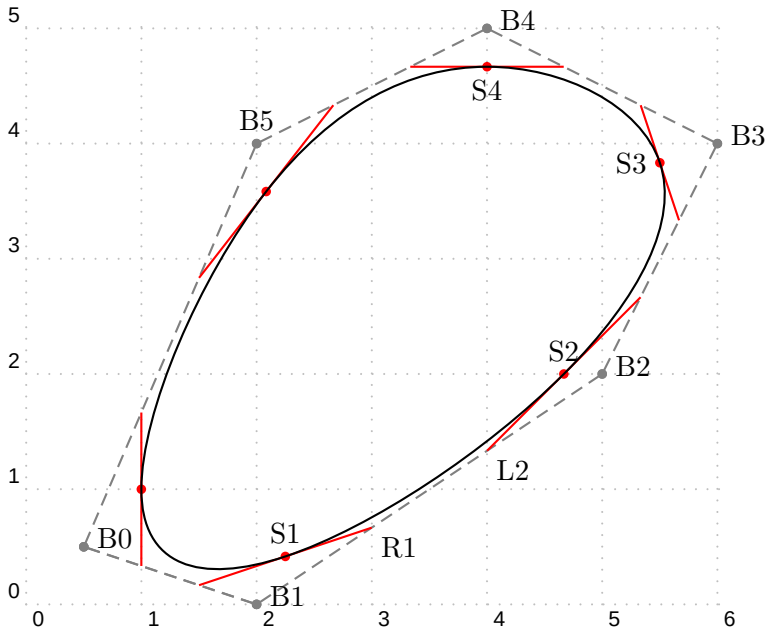
- Extend B_i periodically with period $n+1$, so that $B_{n+1} = B_0$ and $B_{n+2} = B_1$.
- Construct R_i, L_i for $0 < i < n+2$, as above.
- Construct S_k as above (midpoint of L_kR_k), for $0 < k < n+2$.
- Set $S_0 = S_{n+1}$.
- For $0 < k \leq n+1$, construct the cubic Bézier curve with control points S_{k-1}, R_{k-1}, L_k, S_k, parametrized by $k-1 \leq t \leq k$.

The macro in this case is `\psBsplineC`, where the `C` stands for Closed. The code, being implemented as a `\pscustom` object, does not accept the `doubleline` keyword, but does accept, for example,

```
fillstyle=solid,fillcolor=gray
```

```
\documentclass{article}
\usepackage{pstricks}
\usepackage{multido,pst-node,pst-bspline}
\pagestyle{empty}
\begin{document}
\SpecialCoor % essential for pst-bspline package
\psset{unit=.6in}
\begin{pspicture}[showgrid=true](-.5,-.5)(6,5)
\psBsplineC[showframe=true]{B}(.5,.5)(2,0)(5,2)(6,4)(4,5)(2,4)
\multido{\i=0+1}{5}{\uput[20](B\i){B\i}}
\uput[90](B5){B5}\uput[90](BS1){S1}
\uput[90](BS2){S2}\uput[180](BS3){S3}
\uput[270](BS4){S4}\uput[-45](BR1){R1}
```

```
\uput[-45](BL2){L2}
\end{pspicture}
\end{document}
```



4. Related constructions

There are in addition three additional macros that draw similar curves, but organized in a slightly different way. They are particularly useful when there is a sequence of points already defined as `\pnode`s. Here is a simple way to define such a sequence.

4.1. The `pnodes` macro.

The line

```
\pnodes{P}(2,1.5)(3,4)(5,1)
```

defines a sequence of `\pnode`s with the node root `P`: `P0`=(2,1.5), `P1`=(3,4) and `P0`=(5,1). The sequence may be any (reasonable) length. The macro leaves an entry in the console saying that it has defined nodes `P0 .. P2`. The three new macros are:

```
\psBsplineNodes{<node root>}{<top index>}
\psBsplineNodesC{<node root>}{<top index>}
\psBsplineNodesE{<node root>}{<top index>}
```

corresponding to the macros `\psBspline`, `\psBsplineC` and `\psBsplineE`. The difference is that the macros with `Nodes` in the name have as arguments the root node name and the last index, rather than the list of points. For

example, with the above definition of P in force, \psBsplineNodes{P}{2} has exactly the same effect as \psBspline(2,1.5)(3,4)(5,1).

4.2. **The \bspcurvepoints macros.** There are two macros that provide for B-spline curves essentially the same functionality as the \pscurvepoints macro from pstricks-add. (That macro takes as input a parametric curve and constructs as output (at the PostScript level) arrays of data associated with the curve: the positions of points along the curve, the increment from the previous point and a normal vector to the curve. The principal uses for such data are (i) the \pspolylineticks macro from pstricks-add, which allows placement of ticks and other marks along a curve that has been approximated by a polyline; (ii) the \polyIntersections macro from pst-node, which allows you to find the points of intersection of the curve (approximated by a polyline) and an arbitrary line.) Following one of the \psBsplineNodes macros, the macros

```
\bspcurvepoints{<source name>}{<source max index>}{<dest. name>}
\bspcurvepointsE{<source name>}{<source max index>}{<dest. name>}
```

work, in the first case, for a relaxed, uniform B-spline curve, and in the second, for such a curve with its initial and final segments removed, corresponding to the output from \psBsplineE rather than \psBspline. In both cases, you may set the keyword plotpoints (default value: 50) to change the number of sample points on each Bézier component. This will result in the construction of PostScript arrays with indices from 0 to $n =$ num of segments$\times$(plotpoints-1). After running

```
\pnodes{B}(1,2)(3,-1)(4,1)(6,2)% define B0..B3
\psBsplineNodes{B}{3}% draw B-spline with control pts B0..B3
\bspcurvepoints[plotpoints=11]{B}{3}{P}% requires previous line
```

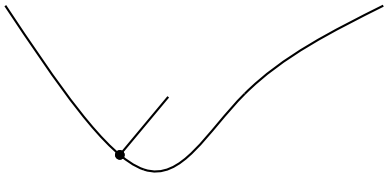
the following PostScript arrays are created, each indexed from 0 to 30:

```
P.X, P.Y (position)
PNormal.X, PNormal.Y (normal vector)
PDelta.X, PDelta.Y (increment from previous position)
```

and these may be used in the usual way to create nodes. For example,

```
\pnode(! P.X 8 get P.Y 8 get ){Q}
\pnode(! PNormal.X 8 get PNormal.Y 8 get ){Dir}
\psrline{*-}(Q)(1cm;{(Dir)})
```

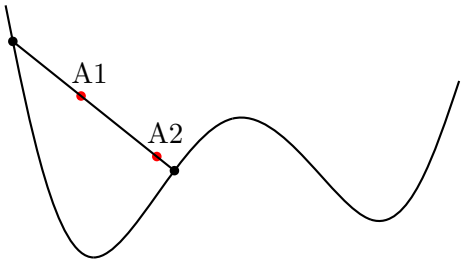
places `Q` at the position on the curve with index 8, defines `Dir` to be a normal vector at that point, then draws a line from `Q` of length `1cm` in the direction



of that normal.

In the next example, we use `\polyIntersections` from `pst-plot` to locate intersections of a line and a B-spline curve.

```
\pspicture(-.5,-.5)(6.5,4.5)%
\pnodes{B}(0,4)(1,-1)(3,4)(5,0)(6,3)% B0..B5
\psBsplineNodes{B}{4}% draw B-spline with control pts B0..B4
\pnode(1,2.8){A1}\pnode(2,2){A2}%
\psdots[linecolor=red](A1)(A2)%
\bspcurvepoints[plotpoints=30]{B}{4}{P}% construct PS arrays,
\bspcurvenodes{P}{Q}% turn them into nodes
% indices 0..116 (=4*29)
\polyIntersections{N1}{N2}(A1)(A2){Q}{116}%
\psline{*-*}(N1)(N2)%
% N1, N2 are points of intersection of curve with A1A2
\Put{;75}(A1){A1}\Put{;75}(A2){A2}
\endpspicture
```



4.3. **Setting nodes on a B-spline curve.** To set a node at t on a B-spline curve after`\bspcurvepoints[E]`, call the macro

```
\bspNode{<control point root>}{<top index>}{<t>}{<node name>}
```

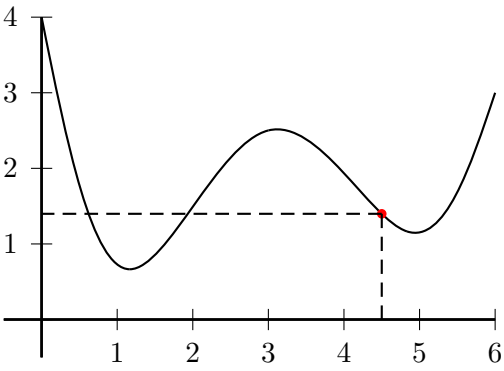
For example, if I have constructed a B-spline curve using control points $B_0,\ldots,B_5$, then `\bspNode{B}{5}{2.1}{Q}` defines a node named Q at $t = 2.1$.

The macro `\bspcurvenodes{P}{R}` creates a node sequence `R0..Rn` at the locations specified by the arrays `P.X`, `P.Y`. (Those arrays must first have been created with one of the `\bspcurvepoints` macros.)

4.4. **B-spline function curves.** By this we mean an open B-spline curve which is the graph of a function $y = f(x)$ and whose orientation is toward the right. It is not analytically simple to specify a formula for f in most cases, and to compute y from x involves (a) finding the index k of the Bézier segment containing x; (b) solving the cubic $x_k(t) = x$ for t, $0 \le t \le 1$; (c) substituting in $y_k(t)$. The package provides a macro to perform these calculations after generating the data using `\bspcurvepoints[E]`:

```
\bspfnNode{<control point root>}{<top index>}{<x0>}{<node name>}
```

```
\pspicture(-.5,-.5)(6.5,4.5)%
\pnodes{B}(0,4)(1,-1)(3,4)(5,0)(6,3)% B0..B4
\psBsplineNodes{B}{4}% draw B-spline with control pts B0..B4
% the curve is graph of a function of x
\bspcurvepoints[plotpoints=10]{B}{4}{P}% construct PS arrays
\bspFnNode{B}{4}{4.5}{QQ}% node QQ on curve at x=4.5
\psdot[linecolor=red](QQ)%
\psline[linestyle=dashed](QQ)(0,0 | QQ)
\psline[linestyle=dashed](QQ)(QQ | 0,0)
\psaxes(0,0)(-.5,-.5)(6,4)
\endpspicture
```



See also the penultimate example in the next section.

5. B-spline Interpolation

This is the inverse problem. Being given points $(S_k)_{0\le k\le n}$, the goal is to produce the B-spline control points B_k leading to the points S_k, so that the associated B-spline curve interpolates the S_k.

5.1. **Open curve.** We discuss first the case of an open, uniform B-spline curve with relaxed endpoints. According to the discussion above, we have to solve the equations

$$\begin{aligned} B_0 &= S_0 \\ B_0 + 4B_1 + B_2 &= 6S_1 \\ B_1 + 4B_2 + B_3 &= 6S_2 \\ &\cdots \\ B_{n-2} + 4B_{n-1} + B_n &= 6S_{n-1} \\ B_n &= S_n \end{aligned}$$

for the B_k. In matrix form, this becomes the tridiagonal system

$$\begin{pmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & 1 & 4 & 1 & \\ & & \cdots & & 1 \\ & & & 1 & 4 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ \cdots \\ B_{n-1} \end{pmatrix} = \begin{pmatrix} 6S_1 - S_0 \\ 6S_2 \\ 6S_3 \\ \cdots \\ 6S_{n-1} - S_n \end{pmatrix}$$

The LU decomposition of the tridiagonal matrix may be seen to take the form

$$\begin{pmatrix} 1 & & & & \\ m_1 & 1 & & & \\ & m_2 & 1 & & \\ & & \cdots & & \\ & & & m_{n-2} & 1 \end{pmatrix} \begin{pmatrix} m_1^{-1} & 1 & & & \\ & m_2^{-1} & 1 & & \\ & & m_3^{-1} & 1 & \\ & & & \cdots & 1 \\ & & & & m_{n-1}^{-1} \end{pmatrix}$$

where $m_1 = 1/4$, $m_{k+1} = 1/(4 - m_k)$ for $k = 1, \cdots, n-2$. The solution of the original system is therefore accomplished in two steps, introducing intermediate points (R_k), by (in pseudo-code)

```
R_1=6*S_1-S_0
for i=2 to n-2
  R_i=6*S_i-m_{i-1}* R_{i-1}
R_{n-1}=(6*S_{n-1}-S_n)-m_{n-2}*R_{n-2}
B_{n-1}=m_{n-1}*R_{n-1}
for i=n-2 downto 1
  B_i=m_i*(R_i-B_{i+1})
```

The code for the `\psBsplineInterp` command uses this algorithm to solve for the B_k as nodes, except that in order to save node memory, the B nodes are substituted in place for the R nodes, so that, for example, the first step becomes `B_1=6*S_1-S_0`.

Assuming you have previously defined nodes `S0` $\cdots$ `S4`,

```
\psBsplineInterp{S}{4}
```

will construct a sequence `SB0` $\cdots$ `SB4` of nodes at the B-spline control points for the relaxed, uniform cubic B-spline interpolating the `Sk`, and this curve may then be rendered with the command

```
\psBsplineNodes{SB}{4}
```

If you don't care about keeping track of the internal operations and names for nodes, you may generate the curve directly with, for example,

```
\psbspline(0,0)(.5,.1)(1.5,.6)(2.5,1.4)(3.5,1.8)(4.5,1.7)%
(5.8,1.0)(7.5,.25)(10,0)
```

5.2. **Closed (periodic) case.** We turn now to the periodic uniform B-spline curve interpolating n points S_0,...,S_{n-1}. Extend the sequence periodically with period n, so that $S_n = S_0$, $S_{n+1} = S_1$, $S_{-1} = S_{n-1}$, and so on. In order to find the periodic control points B_k, we have to solve the n equations

$$\begin{aligned} B_n + 4B_1 + B_2 &= 6S_1 \\ B_1 + 4B_2 + B_3 &= 6S_2 \\ &\cdots \\ B_{n-2} + 4B_{n-1} + B_n &= 6S_{n-1} \\ B_{n-1} + 4B_n + B_1 &= 6S_n \end{aligned}$$

for the B_k, $1 \le k \le n$. In matrix form, this becomes the system

$$\begin{pmatrix} 4 & 1 & & & 1 \\ 1 & 4 & 1 & & \\ & 1 & 4 & 1 & \\ & & \cdots & & 1 \\ 1 & & & 1 & 4 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ \cdots \\ B_n \end{pmatrix} = \begin{pmatrix} 6S_1 \\ 6S_2 \\ 6S_3 \\ \cdots \\ 6S_n \end{pmatrix}$$

Let $(x_k, y_k) = 6S_k$. We perform Gaussian elimination on the matrix

$$\begin{pmatrix} 4 & 1 & & & 1 & x_1 & y_1 \\ 1 & 4 & 1 & & & x_2 & y_2 \\ & 1 & 4 & 1 & & x_3 & y_3 \\ & & \cdots & & 1 & & \\ 1 & & & 1 & 4 & x_n & y_n \end{pmatrix}$$

As in the previous case, let $m_1 = 0.25$, $m_k = 1/(4 - m_{k-1})$ for $k \ge 2$. The factor m_k will be the multiplier of row k after the previous row operation, in order to normalize the row. These are the steps in the procedure.

- Initialize: multiply row 1 by m_1 so that its first entry (1,1) is 1. Replace x_1 by $m_1 x_1$ and y_1 by $m_1 y_1$. Entry $(1, n)$ is m_1.
- Subtract new row 1 from row 2 and multiply the resulting row by m_2. The leading entry (2,1) becomes 1. Entry $(2, n)$ becomes $-m_1 m_2$,

and x_2, y_2 are updated to $m_2(x_2 - x_1)$, $m_2(y_2 - y_1)$. The superdiagonal entry (2,3) is the only other non-zero entry, and its new value is m_2.

- Subtract new row 1 from row n, so that its leading entry $(n, 2)$ is $-m_1$.
- Subtract new row 2 from row 3 and multiply the result by m_3. The leading entry (3,3) becomes 1 and the entry $(3, n)$ becomes $m_1 m_2$, with x_3, y_3 updating to $m_3(x_3 - x_2)$, $m_3(y_3 - y_2)$. The superdiagonal entry (3,4) is now m_3.
- Subtract new row 2 times $-m_1$ from row n, whose leading entry $(n, 3)$ is now $m_1 m_2$.
- Continue in this way until row $n - 2$ has been subtracted as above from row $n - 1$, multiplying the result by m_{n-1}, and a suitable multiple has been subtracted from row n. The leading entry of row $n - 1$ (column $n - 1$) is 1 and its n^{th} entry is $1 - (-1)^n m_1 \cdots m_{n-2}$. Row n has leading entry in column $n - 1$, equal to 1.
- Finally, subtract an appropriate multiple of row $n - 1$ from row n so that row n has leading entry in column n. The resulting matrix is upper triangular, and we may now substitute back starting from the last row to give a complete reduction.

Here are the steps in pseudocode. We keep track of row n with the array b_k, column n with the array c_k. The indices for both run from 1 to n.

```
m(1)=0.25
for k=2 to n-1
   m(k)=1/(4-m(k-1))
b(1)=1
b(n-1)=1
b(n)=4
c(n-1)=1% don't need c(n), =b(n)
%multiply first row by m1
c(1)=m(1)
x(1)=m(1)*x(1)
y(1)=m(1)*y(1)
for k=2 to n-1
  %subtract normalized row k-1 from row k, renormalize row k
  c(k)=m(k)*(c(k)-c(k-1))%note that initially, c(k)=0 for 1<k<n-1
  x(k)=m(k)*(x(k)-x(k-1))
  y(k)=m(k)*(y(k)-y(k-1))
  %subtract normalized row k-1 times b(k-1) from row n
  b(k)=b(k)-b(k-1)*m(k-1)
  b(n)=b(n)-c(k-1)*b(k-1)
```

```
  x(n)=x(n)-x(k-1)*b(k-1)
  y(n)=y(n)-y(k-1)*b(k-1)
% subtract row n-1 times b(n-1) from row n, renormalize by 1/b(n)
b(n)=b(n)-b(n-1)*c(n-1)
x(n)=(x(n)-x(n-1)*b(n-1))/b(n)
y(n)=(y(n)-y(n-1)*b(n-1))/b(n)
%work back
x(n-1)=x(n-1)-c(n-1)*x(n)
y(n-1)=y(n-1)-c(n-1)*y(n)
for k=n-2 downto 1
   x(k)=x(k)-m(k)* x(k+1)-c(k)*x(n)
   y(k)=y(k)-m(k)* y(k+1)-c(k)*y(n)
```

This algorithm is implemented in TEX/PostScript code in `pst-bspline.tex` and may be invoked using the macro

```
\psBsplineInterpC{<node root>}{<index>}
```

You must previously have defined a sequence, say `S0` $\cdots$ `S100` of `\pnode`s that you plan to interpolate with a closed curve. If you used `\pnodes` to do this, it would have constructed a macro `\Snodecount` to store the value 100. Then

```
\psBsplineInterpC{S}{100}
```

constructs the sequence `SB0` $\cdots$ `SB100` of B-spline control points (appending `B` to the root name) for a closed curve interpolating `S0` $\cdots$ `S100`, which may then be rendered with the command

```
\psBsplineNodesC{SB}{101}
```

with any keywords options you wish.

IMPORTANT: The macro `\psBsplineInterpC` modifies the node sequence `S`, setting `S101=S0`, and changing `\Snodecount` to take the value 101. This is convenient when you use the construction:

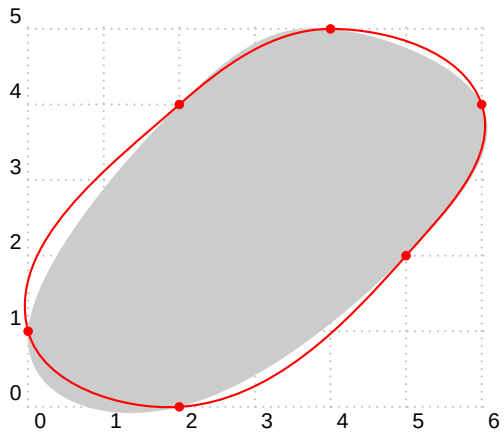
```
\pnodes{S}()()()()% sets \Snodecount to 3
\psBsplineInterpC{S}{\Snodecount}% constructs B-spline control pts SB0..SB4
\psBsplineNodesC{SB}{\Snodecount}
```

The following example illustrates that there is a difference between `\psccurve` and B-spline interpolation, the former having a rounder appearance. Generally speaking, B-spline interpolation comes closer to minimizing the average curvature.

```
\documentclass{article}
\usepackage{pstricks}
\usepackage{pst-bspline,pstricks-add}
\begin{document}
```

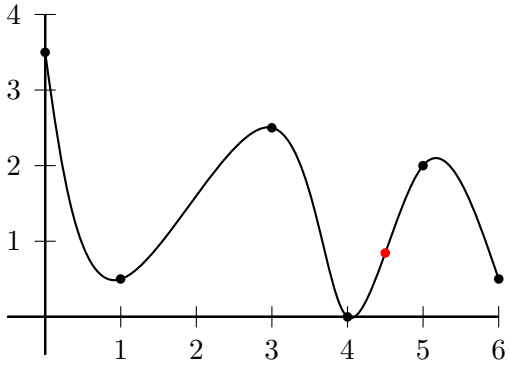
```
\begin{pspicture}[showgrid=true](-.5,-.5)(6,5)
\pnodes{P}(0,1)(2,0)(5,2)(6,4)(4,5)(2,4)%
\psBsplineInterpC{P}{5}%
\psBsplineNodesC*[linecolor=gray!40]{PB}{5}%
\psccurve[linecolor=red,showpoints=true](0,1)(2,0)(5,2)(6,4)(4,5)(2,4)
\end{pspicture}
\end{document}
```

Slight difference between psccurve and B-spline interpolation

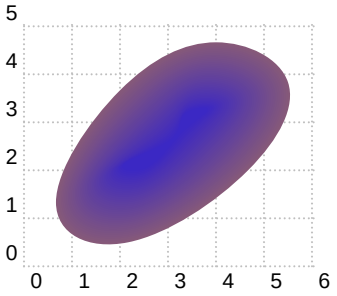


A B-spline curve can in many cases provide a good function interpolation mechanism, but the result is not guaranteed to be the graph of a function.

```
\begin{pspicture}(-.5,-.5)(6,4)
\psdots(0,3.5)(1,.5)(3,2.5)(4,0)(5,2)(6,.5)%
\pnodes{S}(0,3.5)(1,.5)(3,2.5)(4,0)(5,2)(6,.5)% S0..S5
\psBsplineInterp{S}{5}% construct SB0..SB5
\psBsplineNodes{SB}{5}% draw B-spline with control pts SB0..SB5
\bspcurvepoints[plotpoints=10]{SB}{5}{P}
% construct the PS arrays
\bspFnNode{SB}{5}{4.5}{QQ}% node QQ on curve at x=4.5
\psdot[linecolor=red](QQ)%
\psaxes(0,0)(-.5,-.5)(6,4)
\end{pspicture}
```




```
\documentclass{article}
\usepackage{pstricks}
\usepackage{pst-bspline,pstricks-add}
\begin{document}
\psset{unit=.25in}
\begin{pspicture}[showgrid=true](-.5,-.5)(6,5)
\pnodes{P}(0,1)(2,0)(5,2)(6,4)(4,5)(2,4)
\pnode(3,3){C}
\multido{\ra=0+.05,\rb=1+.05,\i=30+1}{40}{%
\psBsplineC*[linecolor=blue!\i!brown]{B}%
([nodesep=\ra]{C}P0)([nodesep=\ra]{C}P1)%
([nodesep=\ra]{C}P2)([nodesep=\ra]{C}P3)%
([nodesep=\ra]{C}P4)([nodesep=\ra]{C}P5)}
\end{pspicture}
\end{document}
```



6. Thick B-spline curves

Inspired by the package `pst-thick`, we provide a similar option for curves generated as B-spline interpolations. The new macro that accomplishes this is

```
\thickBspline#1#2#3#4
%#1=root | #2=nsegments | #3=thickness | #4=items to clip
```

which expects the following data.

- A node sequence. This can be constructed with a command like

  `\pnodes{S}(0,0)(5,1)(4,4)(1,3)%`

  which declares nodes S0..S3, and sets the macro `\Snodecount` to 3.

- An interpolation command, such as

  `\psBsplineInterp{S}{\Snodecount}%`

  which creates a framework of B-spline control points `SB0..SB3`.

- Create the interpolating curve and the Bézier control points for its components, with names like `SBR0..SBR2,SBL1..SBL3` etc, using

  `\psBsplineNodes[linestyle=none]{SB}{\Snodecount}%`

  (The `[linestyle=none]` may be omitted if you want the curve to show.)

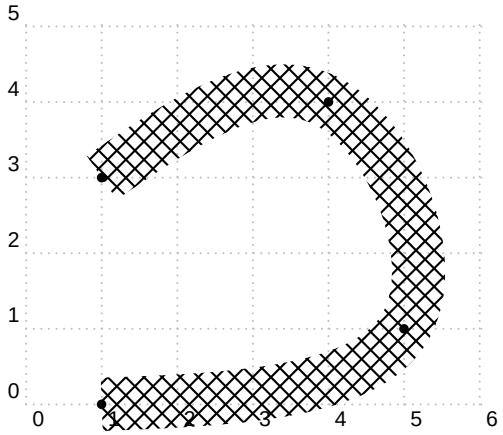
- Create a clipping path of specified thickness around the interpolating curve and place graphics to be clipped:

  ```
  \thickBspline[plotpoints=50,linestyle=none]{S}{3}{20pt}%
  {\psline[linecolor=red,linestyle=solid](0,0)(6,6)}%
  ```

  (The `[linestyle=none]` controls whether the clipping path is rendered, and `plotpoints` controls the number of subdivisions of each Bézier component. Its default value is 50.)

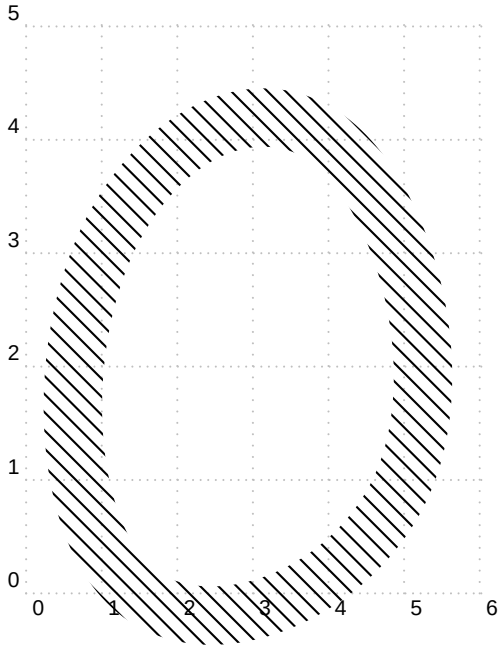
The clipping path is drawn by default positively oriented so that objects are clipped to its interior. By specifying the keyword `reverseclip`, the clipping path will be reversed so that objects are clipped to the exterior.

```
\documentclass{article}
\usepackage{pstricks}
\usepackage{pst-bspline,pstricks-add}
\begin{document}
\begin{pspicture}[showgrid=true](-.5,-.5)(6,5)
\pnodes{S}(1,0)(5,1)(4,4)(1,3)%
\psdots(1,0)(5,1)(4,4)(1,3)%
\psBsplineInterp{S}{\Snodecount}%
\psBsplineNodes[linestyle=none]{SB}{\Snodecount}%
\thickBspline[plotpoints=50,linestyle=none]{S}{3}{20pt}%
{\psframe[fillstyle=crosshatch](-1,-1)(6,6)}%
\end{pspicture}
\end{document}
```



The \thickBspline macro works as expected in the closed (periodic) case, taking advantage of automatic incrementing of the nodecount. Note that \thickBspline interprets thickness as visual, unaffected by possible differences between xunit and yunit.

```
\documentclass{article}
\usepackage{pstricks}
\usepackage{pst-bspline,pstricks-add}
\begin{document}
\psset{yunit=1.5cm}
\begin{pspicture}[showgrid=true](-.5,-.5)(6,5)
\pnodes{S}(1.5,0)(5,1)(4,4)(1,3)%
\psBsplineInterpC{S}{3}%
% defines nodes SB0, SB1, SB2, SB3, SB4 --- the Bspline control points
% increments \Snodecount by 1 for future macros
% Don't use C form of \psBsplineNodes with this new \Snodecount
\psBsplineNodes[linestyle=none,showpoints=false]{SB}{\Snodecount}%
% Constructs the Bezier control points SBR0, SBL1, SBR1, etc
\thickBspline[linestyle=none]{S}{\Snodecount}{22pt}%
{\psframe[fillstyle=vlines](-1,-1)(6,6)}%
\end{pspicture}
\end{document}
```


5
4
3
2
1
0
0
1
2
3
4
5
6