Cubic B-splines Using PSTricks

Michael Sharpe
msharpe@ucsd.edu

A cubic, uniform B-spline curve with control points $B_0 \ldots B_n$ ($n \geq 2$) 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

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

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

\texttt{\textbackslash 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.

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

\texttt{\textbackslash 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.

\texttt{\textbackslash psBsplineNodes\{B\}\{4\}:} draws the relaxed, uniform B-spline with control points $B_0..B_4$.

\texttt{\textbackslash psBsplineNodesE\{B\}\{4\}:} is the same as \texttt{\textbackslash psBsplineNodes\{B\}\{4\}} except that it omits the first and last segments.
\psBsplinNodesC{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 \psBsplinNodesE{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 \psBsplinNodes{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 \psBsplinNodes 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 \psBsplinNodes[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 \( x_0 < x_1 < \cdots \).

\psBsplinInterp{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 \psBsplinNodes{SB}{4}.)

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

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

\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.

\thickBEspline{S}{5}{30pt}{18pt}{-70}{<graphic>}: defines a clipping path of width from 18pt to 30pt and axis angle -70 degrees, then draws the <graphic> clipped to that path. The clipping path is widest where the path is parallel to the axis angle. This macro is best called from the following auxiliary macro, because the data must be prepared in a specific order—see the example provided.
\texttt{\texttt{\textbackslash thickBdraw}\{S\}\{30pt\}\{18pt\}\{-70\}\{\texttt{\textbackslash graphic}\}}: \ \text{prepares and runs the}
\texttt{\texttt{\textbackslash thickBEspline}} \text{using all points in the node sequence } S, \ \text{apassing all parameters to that macro.}

\texttt{\texttt{\textbackslash thickBSplinePen}\{S\}\{5\}\{.7cm\}\{.2cm\}\{-45\}\{\texttt{\textbackslash graphic}\}}: \ \text{defines a clipping path following the outline of an elliptical pen drawn along the}
\text{curve interpolating the points } S_0..S_5. \ \text{The ellipse has } a = .7\text{cm}, \ b = .2\text{cm and and is rotated through -70 degrees. The macro then}
\text{draws the } <\texttt{\textbackslash graphic}> \text{clipped to that path. The underlying curve has}
\text{relaxed endpoints.}

\texttt{\texttt{\textbackslash thickBSplinePenE}\{S\}\{5\}\{.7cm\}\{.2cm\}\{-45\}\{\texttt{\textbackslash graphic}\}}: \ \text{defines a clipping path following the outline of an elliptical pen drawn along the}
\text{curve interpolating the points } S_0..S_5. \ \text{The ellipse has } a = .7\text{cm}, \ b = .2\text{cm and and is rotated through -70 degrees. The macro then}
\text{draws the } <\texttt{\textbackslash graphic}> \text{clipped to that path. The underlying curve}
\text{omits the initial and final segments, and is not limited to relaxed endpoints.}

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 imme-
  diate 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 \texttt{pst-Bspline} package implements this algorithm as \texttt{\textbackslash psBspline}, whose
simplest form is, for example

\texttt{\textbackslash 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 key-
words, like \texttt{linestyle}, \texttt{linecolor} and \texttt{arrows}, there is a Boolean keyword
The effect of \texttt{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,

\begin{verbatim}
\psBspline{B}(.5,.5)(2,0)(5,2)(6,4)(4,5)(2,4)
\end{verbatim}

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

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

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

\begin{verbatim}
\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}(-.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){S1}
\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}
\end{verbatim}
2.1. **Bézier curves as B-spline curves.** Consider a Bézier curve $C$ with control points $S_1, C_1, C_2, S_2$. To identify $C$ 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 \texttt{bsplineE} yields $C$. This is a simple problem in linear algebra whose solution is:
B0=6S1­7C1+2C2  
B1=2C1­C2  
B2=­C1+2C2  
B3=6S2+2C1­7C2

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
\begin{verbatim}
\beztobsp(S1)(C1)(C2)(S2){B}
\end{verbatim}
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 \( L_k S_k \)), 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,
\begin{verbatim}
fillstyle=solid,fillcolor=gray
\end{verbatim}
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 \texttt{\textbackslash pnodes}. Here is a simple way to define such a sequence.

4.1. The \texttt{\textbackslash pnodes} macro. The line
\begin{verbatim}
\texttt{\textbackslash pnodes\{P\}(2,1.5)(3,4)(5,1)}
\end{verbatim}
defines a sequence of \texttt{\textbackslash pnodes} with the node root \texttt{P}: \texttt{P0}=(2,1.5), \texttt{P1}=(3,4) and \texttt{P0}=(5,1). The sequence may be any (reasonable) length. The macro leaves an entry in the console saying that it has defined nodes \texttt{P0} .. \texttt{P2}. The three new macros are:

\begin{verbatim}
\texttt{\textbackslash psBsplineNodes\{<node root>\}\{<top index>\}}
\texttt{\textbackslash psBsplineNodesC\{<node root>\}\{<top index>\}}
\texttt{\textbackslash psBsplineNodesE\{<node root>\}\{<top index>\}}
\end{verbatim}
corresponding to the macros \texttt{\textbackslash psBspline}, \texttt{\textbackslash psBsplineC} and \texttt{\textbackslash psBsplineE}. The difference is that the macros with \texttt{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, $\texttt{\textbackslash psBsplineNodes}\{P\}\{2\}$ has exactly the same effect as $\texttt{\textbackslash psBspline}(2,1.5)(3,4)(5,1)$.

4.2. The $\texttt{\textbackslash bspcurvepoints}$ macros. There are two macros that provide for B-spline curves essentially the same functionality as the $\texttt{\textbackslash pscurvepoints}$ macro from $\texttt{pstricks-add}$. (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 $\texttt{\textbackslash pspolylineticks}$ macro from $\texttt{pstricks-add}$, which allows placement of ticks and other marks along a curve that has been approximated by a polyline; (ii) the $\texttt{\textbackslash polyIntersections}$ macro from $\texttt{pst-node}$, which allows you to find the points of intersection of the curve (approximated by a polyline) and an arbitrary line.) Following one of the $\texttt{\textbackslash psBsplineNodes}$ macros, the macros

\begin{verbatim}
\texttt{\textbackslash bspcurvepoints\{<source name>\}\{<source max index>\}\{<dest. name>\}}
\texttt{\textbackslash bspcurvepointsE\{<source name>\}\{<source max index>\}\{<dest. name>\}}
\end{verbatim}

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 $\texttt{\textbackslash psBsplineE}$ rather than $\texttt{\textbackslash psBspline}$. In both cases, you may set the keyword $\texttt{plotpoints}$ (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 $n = \text{num of segments} \times \text{plotpoints}$-1. After running

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

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,

\begin{verbatim}
\texttt{\textbackslash pnode(\{}P.X \text{8 get} \text{P.Y \text{8 get}}\}\{Q\}}
\texttt{\textbackslash pnode(\{}PNormal.X \text{8 get} \text{PNormal.Y \text{8 get}}\}\{Dir\}}
\texttt{\textbackslash psrline(\star\text{-}\}\{Q\}(1\text{cm};\{(Dir)\})}
\end{verbatim}
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 \texttt{polyIntersections} from \texttt{pst-plot} to locate intersections of a line and a B-spline curve.

\begin{verbatim}
\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,
\bspcurvnenodes{P}{Q}% turn them into nodes
% indices 0..116 (=4*29)
\polyIntersections{N1}{N2}{A1}{A2}{Q}{116}%
\psline{*-*}(N1)(N2)%
\Put{;75}(A1){A1}\Put{;75}(A2){A2}\
\endpspicture
\end{verbatim}

4.3. Setting nodes on a B-spline curve. To set a node at \(t\) on a B-spline curve after \texttt{\textbackslash bspcurvepoints[E]}, call the macro

\texttt{\textbackslash 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 \texttt{\textbackslash bspNode\{B\}{5}{2.1}\{Q\}} defines a node named Q at \(t = 2.1\).
The macro \texttt{\b npc r u v e nodes\{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 \texttt{\b np c r u v e points} macros.)

4.4. \textbf{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 \( t, 0 \leq t \leq 1 \); (c) substituting in \( y_k(t) \). The package provides a macro to perform these calculations after generating the data using \texttt{\b np c r u v e points}[E]:

\begin{verbatim}
\bspfnNode{<control point root>}{<top index>}{<x0>}{<node name>}
\end{verbatim}

See also the penultimate example in the next section.

5. \textbf{B-spline Interpolation}

This is the inverse problem. Being given points \((S_k)_{0 \leq k \leq n}\), 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

\[
B_0 = S_0 \\
B_0 + 4B_1 + B_2 = 6S_1 \\
B_1 + 4B_2 + B_3 = 6S_2 \\
\vdots \\
B_{n-2} + 4B_{n-1} + B_n = 6S_{n-1} \\
B_n = S_n
\]

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

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

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

\[
\begin{pmatrix}
1 & & & & \\
\frac{1}{m_1} & 1 & & & \\
\frac{1}{m_2} & \frac{1}{m_1} & 1 & & \\
\vdots & \ddots & \ddots & \ddots & \\
\frac{1}{m_{n-2}} & & \frac{1}{m_{n-3}} & \cdots & 1
\end{pmatrix}
= \begin{pmatrix}
1 & & & & \\
\frac{1}{m_1} & 1 & & & \\
\frac{1}{m_2} & \frac{1}{m_1} & 1 & & \\
\vdots & \ddots & \ddots & \ddots & \\
\frac{1}{m_{n-2}} & & \frac{1}{m_{n-3}} & \cdots & 1
\end{pmatrix}
\]

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

```latex
R_{-1} = 6 \ast S_1 - S_0 \\
\text{for } i = 2 \text{ to } n-2 \\
\quad R_i = 6 \ast S_i - m_{\{i-1\}} \ast R_{\{i-1\}} \\
\quad R_{\{n-1\}} = (6 \ast S_{\{n-1\}} - S_n) - m_{\{n-2\}} \ast R_{\{n-2\}} \\
B_{\{n-1\}} = m_{\{n-1\}} \ast R_{\{n-1\}} \\
\text{for } i = n-2 \text{ downto } 1 \\
\quad B_i = \frac{m_i}{R_i - B_{\{i+1\}}}
```

The code for the \texttt{\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 \(B_{\{1\}} = 6 \ast S_{\{1\}} - S_0\).

Assuming you have previously defined nodes \(S_0 \cdots S_4\),

\texttt{\psBsplineInterp\{S\}\{4\}}
will construct a sequence \texttt{SB0} \cdots \texttt{SB4} of nodes at the B-spline control points for the relaxed, uniform cubic B-spline interpolating the \(S_k\), and this curve may then be rendered with the command\[ \backslash \text{psBsplineNodes}\{\texttt{SB}\}\{4\} \]

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,

\[
\backslash \text{psbspline}(0,0)(.5,.1)(1.5,.6)(2.5,1.4)(3.5,1.8)(4.5,1.7)\%
(5.8,1.0)(7.5,.25)(10,0)
\]

5.2. **Closed (periodic) case.** We turn now to the periodic uniform B-spline curve interpolating \(n\) points \(S_0,\ldots,S_{n-1}\). 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

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

for the \(B_k\), \(1 \leq k \leq n\). In matrix form, this becomes the system

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

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

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

As in the previous case, let \(m_1 = 0.25\), \(m_k = 1/(4 - m_{k-1})\) for \(k \geq 2\). 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 \(m_1x_1\) and \(y_1\) by \(m_1y_1\). 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 \(-m_1m_2\),
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 $m_1m_2$, 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 $m_1m_2$.

- 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 $n$.

```plaintext
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)
```


\begin{verbatim}
x(n)=x(n)\cdot x(k-1)*b(k-1) 
y(n)=y(n)\cdot 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 
 \quad x(k)=x(k)-m(k)* x(k+1)-c(k)*x(n) 
 \quad y(k)=y(k)-m(k)* y(k+1)-c(k)*y(n) 
\end{verbatim}

This algorithm is implemented in \TeX/PostScript code in \texttt{pst-bspline.tex}
and may be invoked using the macro

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

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

\texttt{\psBsplineInterpC\{S\}{100}}

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

\texttt{\psBsplineNodesC\{SB\}{101}}

with any keywords options you wish.

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

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

The following example illustrates that there is a difference between \texttt{\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\}
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.
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
  \[\texttt{\textbackslash pnodes}\{S\}(0,0)(5,1)(4,4)(1,3)\%
  \]
  which declares nodes S0..S3, and sets the macro \texttt{\textbackslash Snodecount} to 3.

- An interpolation command, such as
  \[\texttt{\textbackslash psBsplineInterp}\{S\}\{\textbackslash Snodecount\}\%
  \]
  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
  \[\texttt{\textbackslash psBsplineNodes}\{linestyle=none\}\{SB\}\{\textbackslash Snodecount\}\%
  \]
  (The \texttt{\textbackslash 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:
  \[\texttt{\textbackslash thickBspline}\{plotpoints=50,linestyle=none\}\{S\}\{3\}\{20pt\}\%
  \{\texttt{\textbackslash psline}\{linecolor=red,linestyle=solid\}\(0,0\)(6,6)\%
  \]
  (The \texttt{\textbackslash linestyle=solid} controls whether the clipping path is rendered, and \texttt{\textbackslash 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 \texttt{reverseclip}, the clipping path will be reversed so that objects are clipped to the exterior.

\[\texttt{\documentclass\{article\}}\%
\texttt{\usepackage\{pstricks\}}\%
\texttt{\usepackage\{pst-bspline,pstricks-add\}}\%
\begin\{document\}\%
\begin\{pspicture\}[showgrid=true](-.5,-.5)(6,5)\%
\texttt{\textbackslash pnodes}\{S\}(1,0)(5,1)(4,4)(1,3)\%
\texttt{\textbackslash psdots}(1,0)(5,1)(4,4)(1,3)\%
\texttt{\textbackslash psBsplineInterp}\{S\}\{\textbackslash Snodecount\}\%
\texttt{\textbackslash psBsplineNodes}\{linestyle=none\}\{SB\}\{\textbackslash Snodecount\}\%
\texttt{\textbackslash thickBspline}\{plotpoints=50,linestyle=none\}\{S\}\{3\}\{20pt\}\%
\{\texttt{\textbackslash psframe}\{fillstyle=crosshatch\}\(-1,-1\)(6,6)\%
\end\{pspicture\}\%
\end\{document\}
The `\texttt{thickBspline}` macro works as expected in the closed (periodic) case, taking advantage of automatic incrementing of the node count. Note that `\texttt{thickBspline}` interprets thickness as visual, unaffected by possible differences between `xunit` and `yunit`.

\begin{verbatim}
\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}\psBsplineNodes[linestyle=none,showpoints=false]{SB}{3}\psframe[fillstyle=vlines](-1,-1)(6,6)
\end{pspicture}
\end{document}
\end{verbatim}
There are three variants of the \texttt{\texttt{thickBspline}} macro. The first is

\texttt{\texttt{thickBEspline}\#1\#2\#3\#4\#5\#6}
\texttt{\% \#1=root | \#2=nsegments | \#3=maxthickness |}
\texttt{\% \#4=minthickness | \#5=axis angle | \#6=items to clip}

works just like \texttt{\texttt{thickBspline}}, but instead of providing a thickness, you provide three parameters specifying maxthickness, minthickness and an axis angle. An axis angle of -45 means that lines normal to angle -45 will be minthickness and lines normal to 45 will be maxthickness. You may get unexpected results if your initial curve is too curvy. This macro is best called from \texttt{\texttt{thickBdraw}}, as shown in the following example.

\begin{verbatim}
\psset{yunit=.7cm}
\begin{pspicture}[showgrid=false](-1,-1)(6,7)
\psgrid%
\pnodes{S}(0,1)(0,5)(3,5)(4,2)(5,0)%
\thickBdraw{S}{30pt}{16pt}{-70}\psframe[fillstyle=solid,fillcolor=yellow](-1,-1)(6,7)%
\end{pspicture}
\end{verbatim}
A second variant \texttt{\thickBsplinePen} simulates the effect of an elliptical pen run along the specified curve. Note that you will receive an error message if \texttt{xunit} differs from \texttt{yunit}—the calculations in the general case are quite nasty. The macro is called with something like

\begin{verbatim}
\thickBsplinePen{\S}{\Snodecount}{2cm}{.5cm}{10}{% 
\psframe[fillstyle=crosshatch](-1,-1)(7,6)}%
\end{verbatim}

where \texttt{\S} is an established node sequence with highest index \texttt{\Snodecount} and the pen has semi-major axis 2cm, semi-minor axis .5cm, and the major axis is rotated 10 degrees from the positive x axis. This gives results that look like those generated in \texttt{metafont/post} except that the curve ends are not drawn by the pen—if you want them, draw them separately using \texttt{\psellipse}. Note that the underlying curve has relaxed ends.

\begin{verbatim}
\begin{pspicture}[showgrid=false](-1,-1)(7,7)
\grid
\nodes{\S}(1,0)(4,1)(5,2)(3,3)(1,5)(5,6)%
\thickBsplinePen[linecolor=green]{\S}{\Snodecount}{.7cm}{.1cm}{-45}{% 
\psframe[fillstyle=solid](-1,-1)(7,6)}% 
\psBsplineNodes[linestyle=solid]{\S}{\Snodecount}%
\end{pspicture}
\end{verbatim}
A third variant \texttt{\textbackslash thickBsplinePenE} is quite similar but omits the first and last segments of the underlying curve, permitting the endpoints to not be relaxed.

\begin{pspicture}[-1,0](7,7)
\psgrid
\pnodes{S}(0,4)(1,0)(4,.5)(5,2)(3,3)(1,5)(5,6)(5,3)
\thickBsplinePenE[linecolor=green]{S}{\Snodecount}{.7cm}{.1cm}{30}{\psframe[fillstyle=solid,linestyle=solid]{SB}{\Snodecount}}
\end{pspicture}