Splines and curves in grid

December 20, 2005

From R version 2.3.0, a new graphical primitive is available: the **X-spline** (Blanc and Schlick, 1995). An X-spline is a parameterised curve through a set of control points. A single 'shape' parameter allows the curve to vary from an approximating cubic B-spline to an interpolating Catmull-Rom spline. The spline may be open or closed; if it is open, either end can be adorned with an arrow, and if it is closed, it can be filled with a colour.

Simple examples of X-splines through four control points, for various 'shape's at each control point, are produced by the following code and shown in figure 1.

```
> example(grid.xspline)
```

X-splines can be used to easily draw exciting curvy lines instead of boring straight lines, for example, when labelling a point of interest in a plot, as in Figure 2.

Another use for closed X-splines is the creation of arbitrary smooth shapes (c.f., the use of the polygon primitive for arbitrary "pointy" shapes). Figure 3 shows a couple of examples (with the corresponding polygon in grey).

Curves

Another new primitive, the **curve** has been developed that uses X-splines to draw smooth curves between points. Given two locations (or several pairs of locations), this function can be used to draw a nice curve from one location to the other. Here is a simple example (see Figure 4):

```
> x1 <- unit(0.2, "npc")
> y1 <- unit(0.15, "npc")
> x2 <- unit(0.8, "npc")
```

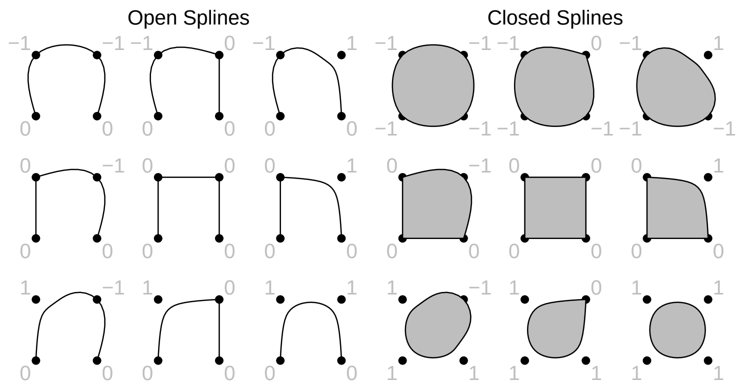


Figure 1: Some simple X-splines through four control points.

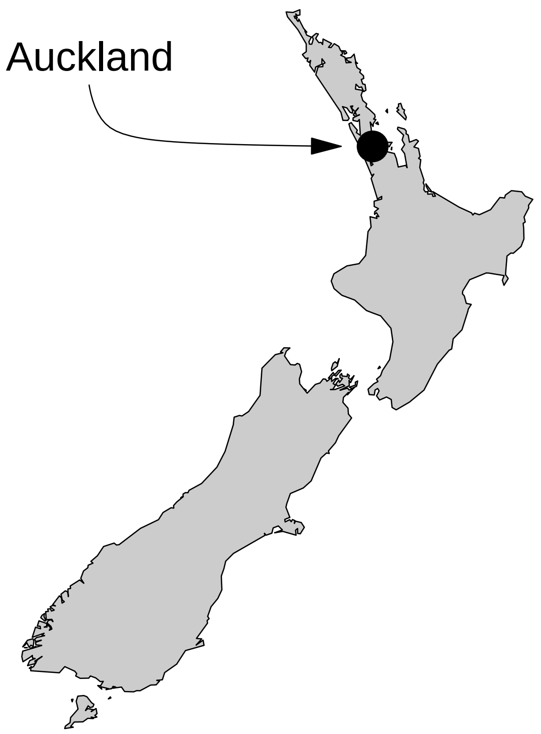


Figure 2: A simple use of an X-spline for labelling a plot.

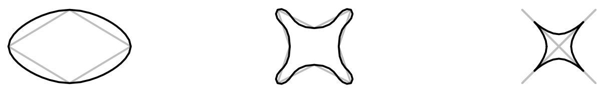


Figure 3: Some amusing shapes produced using X-splines.

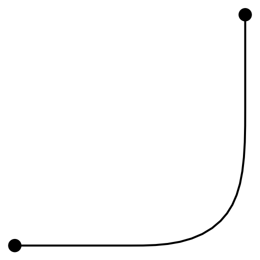


Figure 4: A simple curve between two points.

```
> y2 <- unit(0.75, "npc")
> grid.points(x1, y1, pch = 16, size = unit(2, "mm"))
> grid.points(x2, y2, pch = 16, size = unit(2, "mm"))
> grid.curve(x1, y1, x2, y2)
```

The curve that `grid.curve()` draws is an X-spline so it depends on a set of control points. The locations of the control points are calculated automatically based on a circle that intersects the end points. Figure 5 shows the curve from Figure 4 with a grey circle added to show the circle that was used to produce the control point; a blue point has been added to show the control point itself. Further markings show the origin of the circle (a small grey dot; the larger grey dot will be explained later) and distinguish the start and end points by colouring them green and red respectively.

There are several arguments that provide many ways to modify the basic curve:

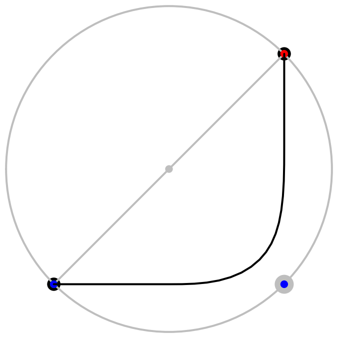
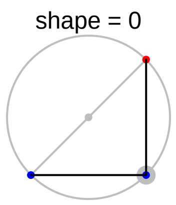
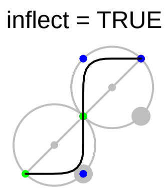
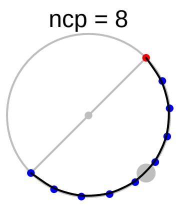
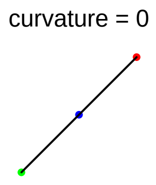


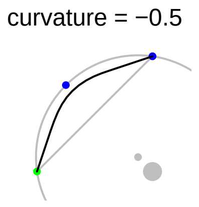
Figure 5: The simple curve from Figure 4 with debugging information added.

`shape`	This argument controls the shape of the X-spline at each control point. For example, using `shape=0` means that the curve is discontinuous at the control point.	
`inflect`	This argument allows the curve to be repeated and inverted to produce a point of inflection.	
`ncp`	This argument controls the number of control points that are generated; this basically controls the smoothness of the curve. Using eight or more control points produces a very smooth curve. As the number of control points increases, the influence of the `shape` argument is greatly reduced.	

`curvature`

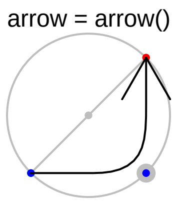
This argument controls the distance of the control points from the straight line joining the end points. The sign of this argument is important: a positive value means that the control points are to the right of a straight line looking from the start point to the end point; a negative value means that the control points are to the left of a straight line. A curvature of 0 produces a straight line.
Another way to think about this argument is that is controls the origin of the circle that is used to calculate the control points. The default curvature is 1, which means that the origin of the circle lies on the line connecting the end points; the control points lie on a semi-circle, with origin half-way between the end points. a curvature of -0.5 means that the origin of the circle is to the right of the line connecting the end points, so the curve is a "flatter", left-hand curve.





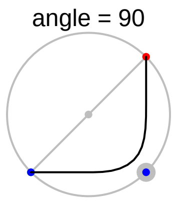
`arrow`

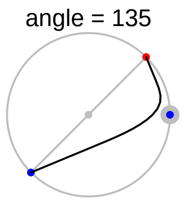
This argument controls whether arrows are drawn at either end of the line. A non-`NULL` value should be provided using the `arrow()` function, which allows for arrows at both ends of the line, allows the arrows to be closed, and so on.



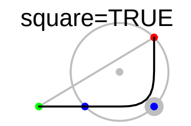
`angle`

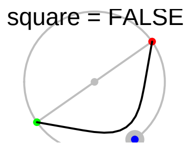
This argument is a subtle one. It controls the placement of control points by allowing them to be skewed towards one or other end point. What happens is that a "corner" is calculated from the end points so that the angle between the start control point and corner is `angle` (the corner is shown as a large grey dot in Figure 5). The circle that is used to generate control points is skewed so that the midpoint of the arc on this circle maps to the corner. The default value of `angle` is 90, which means that the control points are not skewed at all.





`square` This argument is also a bit subtle. If it is `TRUE`, then an additional control points is automatically added so that there is a 45-degree line between that control point and the start point or end point. The effect of this is that the default curve is drawn between the end points in a city-block fashion, left-right and then up-down, or vice-versa.
If this argument is `FALSE`, no additional control points are added.





Vectorised arguments

It is possible to draw multiple curves from a single call to `grid.curve()` by specifying a vector of values for either end point. The `curvature`, `angle`, `ncp`, and `inflect` arguments are all single-valued so multiple curves will all have the same fundamental shape. The `shape` argument can be a vector, but this is simply to allow fine control over each control point when there is more than one control point (i.e., the same vector of shape values is applied to all curves drawn by a single call to `grid.curve()`). The `arrow` and `gp` arguments may contain vector values, in which case each curve may receive different values.

The following code demonstrates these rules by drawing three curves, all originating from the same location, but ending at three different locations. Each curve has a different style of arrow head, and a different fill colour applied to it (see Figure 6).

```
> x3 <- unit(0.1, "npc")
> y3 <- unit(0.1, "npc")
> x4 <- unit(2:4/5, "npc")
> y4 <- unit(0.9, "npc")
> grid.curve(x3, y3, x4, y4, arrow = arrow(type = c("open", "closed",
+     "closed"), angle = c(15, 30, 45), length = unit(3:1/10, "inches")),
+     gp = gpar(fill = c(NA, "grey", "black")))
```

More examples

This section describes some useful combinations of the various `grid.curve()` arguments.

loops By defining end points very close together, a very large curvature, and

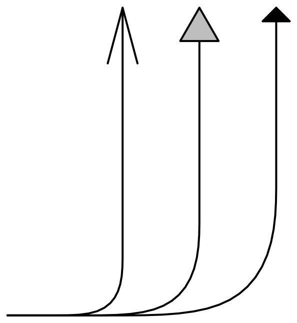
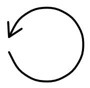


Figure 6: Multiple curves from a single `grid.curve()` call.

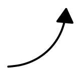
lots of control points, it is possible to produce a loop pointing back on itself.

```
curvature=10, ncp=10,
square=FALSE, arrow=arrow(...)
```



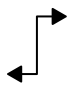
arcs If you want to produce a curve between two points that represents an arc, there is an `arcCurvature()` function that calculates the correct curvature given an angle for the arc. A simple example is to draw a quarter circle between two points. Using a large number of control points ensures the curve follows the arc on a circle; for general use, it is necessary to specify `square=FALSE`.

```
curvature=arcCurvature(90),
ncp=8, square=FALSE,
arrow=arrow(...)
```



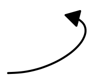
connectors Two-way arrows between locations in a grid are simple using the `inflect` argument, a single control point, and a `shape` of zero.

```
shape=0, inflect=TRUE,
arrow=arrow(...)
```



overshoot Curves that shoot past the destination and curve back around are possible with a large number of control points, and variation of the `angle` argument. You will probably have to use `square=FALSE` to produce a nice result.

```
shape=-1, angle=130,
square=FALSE,
ncp=8, arrow=arrow(...)
```



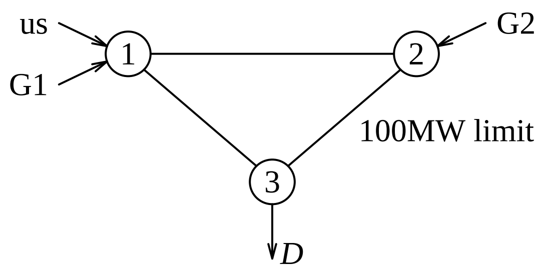
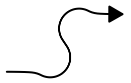


Figure 7: A node-and-edge graph.

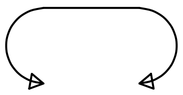
curly ones The default "square" curves work well with a `shape` of zero or less, a single control point, and the default `angle` of 90. Variations on these settings can produce some interesting curves.

```
squareShape=1, inflect=TRUE,
ncp=3, arrow=arrow(...)
```



general paths By combining X-splines and other line-drawing primitives, it is possible to produce arbitrary paths.

```
grid.curve(x1, y1, x1, y2, ...)
grid.lines(x1, y2, x2, y2, ...)
grid.curve(x2, y2, x2, y1, ...)
```



Node-and-Edge Graphs and Flow Diagrams

Combining these curve-drawing primitives with the ability to calculate the borders of a grob, makes it straightforward to produce node-and-edge graphs and flow diagrams (see the end of this document for code used to produce these figures).

References

R. H. Bartels, J. C. Beatty, and B. A. Barsky. *An introduction to splines for use in computer graphics and geometric modeling*. Kaufmann, 1987.

C. Blanc and C. Schlick. X-splines : A spline model designed for the end user. *Proceedings of SIGGRAPH '95*, 1995.

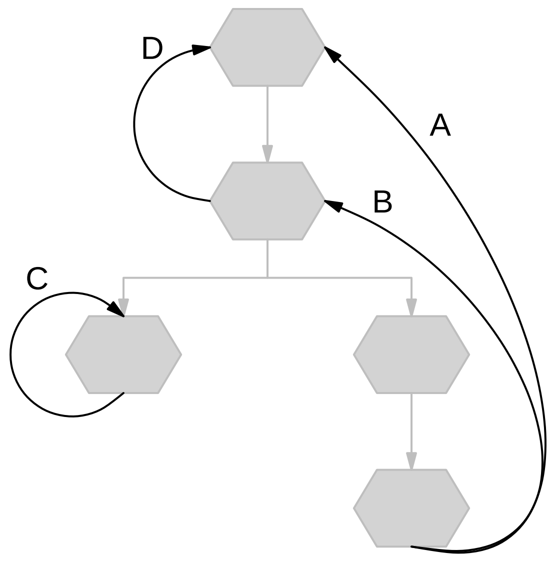


Figure 8: A flow diagram.

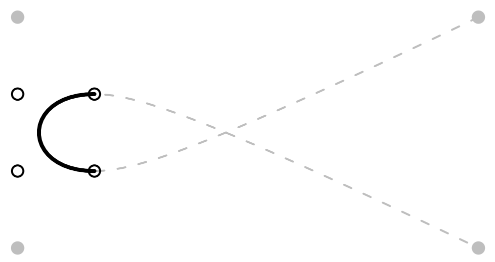


Figure 9: An approximate Bezier curve. The black line (and open circles) is the Bezier curve (and its control points) that we are trying to draw. The grey dashed line (and grey circles) is the B-spline (and its control points) that we use to approximate the Bezier curve.

A Note on Bezier Curves

A (cubic) Bezier curve is a curve described by four control points. The curve goes through the first and last control points and approximates the second and third control points. A Bezier curve with control points $b = (b1, b2, b3, b4)$ is equivalent to a B-spline curve with control points $s = (s1, s2, s3, s4)$ (Bartels et al., 1987), where ...

$$s = Ms^{-1} * Mb * b,$$

$$Ms = \frac{1}{6} * \begin{pmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}, \quad Mb = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}.$$

The (open) curves drawn by default by `grid.xspline()` automatically replicate the first and last control point (so that the curve is drawn from the first and last control point). This can be turned off via the `repEnds` argument and then, using the above formulae, an approximate bezier curve can be obtained.

Figure 9 illustrates this process. In the Figure, the Bezier control points are drawn as open circles, the corresponding B-spline control points are drawn as grey circles, the default output from `grid.xspline()` is shown as a dashed grey line, and the approximation to the desired Bezier curve, produced by setting `repEnds=FALSE`, is drawn as a thick black line. The curve is only approximate because an X-spline with `shape=1` is only approximately a B-spline; but the approximation is quite good.

Code for node-and-edge graph

```
x <- unit(c(0, 1, 0.5), "native")
y <- unit(c(1, 1, 0.5), "native")
grid.text(1:3, x, y)
grid.circle(x[1], y[1], r=unit(0.7, "char"), name="c1")
grid.circle(x[2], y[2], r=unit(0.7, "char"), name="c2")
grid.circle(x[3], y[3], r=unit(0.7, "char"), name="c3")
grid.segments(grobX("c1", "east"), grobY("c1", "east"),
              grobX("c2", "west"), grobY("c2", "west"))
grid.segments(grobX("c1", 315), grobY("c1", 315),
              grobX("c3", 135), grobY("c3", 135))
grid.segments(grobX("c2", 225), grobY("c2", 225),
              grobX("c3", 45), grobY("c3", 45))
arr <- arrow(angle=15, length=unit(2, "mm"))
grid.text("us ", -0.25, 1.12, just="right", default="native")
grid.segments(-.24, 1.12, grobX("c1", 160), grobY("c1", 160),
              default="native", arrow=arr)
grid.text("G1 ", -0.25, 0.88, just="right", default="native")
grid.segments(-.24, 0.88, grobX("c1", 200), grobY("c1", 200),
              default="native", arrow=arr)
grid.text(" G2", 1.25, 1.12, just="left", default="native")
grid.segments(1.24, 1.12, grobX("c2", 20), grobY("c2", 20),
              default="native", arrow=arr)
grid.text(" D", 0.5, 0.22, just="left", default="native",
          gp=gpar(fontface="italic"))
grid.segments(grobX("c3", 270), grobY("c3", 270), 0.5, 0.2,
              default="native", arrow=arr)
grid.text("100MW limit", 0.8, 0.7, just="left", default="native")
```

Code for flow diagram

```
hex <- function(x, y, width=.2, height=.1, units="npc",
                gp=gpar(col="grey", fill="light grey"), ...) {
    xx <- unit(c(x - 0.3*width, x - 0.5*width, x - 0.3*width,
                 x + 0.3*width, x + 0.5*width, x + 0.3*width),
               units)
    yy <- unit(c(y - 0.5*height, y, y + 0.5*height,
                 y + 0.5*height, y, y - 0.5*height),
               units)
    grid.polygon(xx, yy, gp=gp, ...)
}
# grid.newpage()
```

```
arr <- arrow(angle=15, type="closed", length=unit(0.03, "npc"))
hex(.5, .8, name="h1")
hex(.5, .6, name="h2")
hex(.25, .4, name="h3")
hex(.75, .4, name="h4")
hex(.75, .2, name="h5")
grid.segments(grobX("h1", 270), grobY("h1", 270),
              grobX("h2", 90), grobY("h2", 90),
              arrow=arr, gp=gpar(fill="grey", col="grey"))
grid.curve(grobX("h2", 270), grobY("h2", 270),
           grobX("h3", 90), grobY("h3", 90),
           inflect=TRUE, shape=0, curvature=-1,
           arrow=arr, gp=gpar(fill="grey", col="grey"))
grid.curve(grobX("h2", 270), grobY("h2", 270),
           grobX("h4", 90), grobY("h4", 90),
           inflect=TRUE, shape=0,
           arrow=arr, gp=gpar(fill="grey", col="grey"))
grid.segments(grobX("h4", 270), grobY("h4", 270),
              grobX("h5", 90), grobY("h5", 90),
              arrow=arr, gp=gpar(fill="grey", col="grey"))
grid.curve(grobX("h2", 180), grobY("h2", 180),
           grobX("h1", 180), grobY("h1", 180),
           shape=1, ncp=10, square=FALSE, curvature=-1,
           arrow=arr, gp=gpar(fill="black"))
grid.curve(grobX("h3", 270), grobY("h3", 270),
           grobX("h3", 90), grobY("h3", 90),
           shape=1, square=FALSE, curvature=-3, ncp=10,
           arrow=arr, gp=gpar(fill="black"))
grid.curve(grobX("h5", 270), grobY("h5", 270),
           grobX("h2", 0), grobY("h2", 0),
           shape=1, square=FALSE, curvature=1, ncp=10, angle=60,
           arrow=arr, gp=gpar(fill="black"))
grid.curve(grobX("h5", 270), grobY("h5", 270),
           grobX("h1", 0), grobY("h1", 0),
           shape=1, square=FALSE, curvature=0.8, ncp=10, angle=50,
           arrow=arr, gp=gpar(fill="black"))
grid.text("A", 0.8, 0.7)
grid.text("B", 0.7, 0.6)
grid.text("C", 0.1, 0.5)
grid.text("D", 0.3, 0.8)
```