The `bpolynomial.mp` package*

Stephan Hennig†

November 26, 2007

Abstract

The MetaPost package `bpolynomial.mp` helps drawing polynomial functions of up to degree three. It provides macros to calculate Bézier curves exactly matching a given constant, linear, quadratic or cubic polynomial.

Contents

1 Introduction

MetaPost has a variable type `path` that can be used for drawing smooth and visualy pleasing curves. Internally, paths are Bézier curves and MetaPost is able to calculate the points along such a curve.[1]

When drawing graphs, the problem users are confronted with is how to define a suitable path representing a given function $f(x)$? The `splines` package by Dan Luecking provides macros to draw smooth piece-wise Bézier curves through arbitrary sample points. [2] However, since Bézier curves are polynomials of degree three, we can do better with just one Bézier curve segment for such polynomials. This package eases the task of finding a Bézier curve matching a given polynomial

$$f(x) = ax^3 + bx^2 + cx + d \tag{1}$$

*This document describes `bpolynomial.mp` v0.3, last revised 11/26/2007.

†stephanhennig@arcor.de

[1] Since PostScript has a concept of Bézier curves, too, for MetaPost drawing a path is simply an act of copying the parameters of the corresponding Bézier curve into PostScript output. But nonetheless MetaPost *can* calculate points on a Bézier curve.

2 Usage

2.1 Macro `newBPolynomial`

The `bpolynomial.mp` package provides just one macro `newBPolynomial`. This macro takes one suffix parameter and four numeric parameters that are the coefficients of the given polynomial. A polynomial definition for a function

$$f(x) = 2x^3 + 0x^2 - 3x - 1 \tag{2}$$

exemplary looks like this

```
newBPolynomial.f(2, 0, -3, -1);
```

Here, suffix parameter `f` serves as an identifier where some names of macros and variables, that have to be called later, are derived from and the parameters 2, 0, -3, -1 match the coefficients of our function f. To be more precise, command

`newBPolynomial.<suffix>()`

defines two new macros

`<suffix>.getPath()`

and

`<suffix>.eval()`

that do the real work.

2.2 Macro `<suffix>.getPath`

Macro `<suffix>.getPath(xmin, xmax)` returns a path exactly matching the polynomial defined by `newBPolynomial.<suffix>` on the intervall $[xmin, xmax]$. Let's have a look at an example. Drawing our polynomial $f(x)$ on the intervall $(-2, 2)$ can be done with the following code (figure 1).

```
newBPolynomial.f(2, 0, -3, -1);
draw f.getPath(-2, 2) xscaled 1cm yscaled 0.1cm;
```

Once a polynomial ⟨suffix⟩ has been defined `<suffix>.getPath` can be called as often as required with varying arguments and returns a path corresponding to the requested section of polynomial ⟨suffix⟩.

Note, since the `bpolynomial.mp` package never uses ⟨suffix⟩ as a complete identifier, you can use that as the name of a path variable to store the path returned by `<suffix>.getPath` for later drawing. Any other path (array) variable serves the same purpose, though.

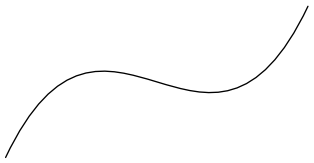

Figure 1: A cubic polynomial.

Figure 2: With stars.

```
newBPolynomial.f(2, 0, -3, -1);
path f;
f := f.getPath(-2, 2);
draw f xscaled 1cm yscaled 0.1cm;
```

2.3 Macro `<suffix>.eval`

The other macro defined by `newBPolynomial.<suffix>`, macro `<suffix>.eval`, can be used to evaluate polynomial ⟨suffix⟩ at a given x-coordinate. This macro takes one parameter—the x-coordinate. A "starred" version of our polynomial can be plotted with the following code (figure 2).

```
newBPolynomial.f(2, 0, -3, -1);
for x=-2 step .25 until 2:
   label(btex $\star$ etex, (x, f.eval(x)) xscaled 1cm yscaled 0.1cm);
endfor
```

2.4 Accessing polynomial coefficients

Additionally, macro `newBPolynomial.<suffix>` saves the coefficients passed as arguments in variables `<suffix>.a`, `<suffix>.b`, `<suffix>.c` and `<suffix>.d` for later reference.

3 Examples

In the first example a simple corrdinate system is drawn manually. Then a quadratic polynomial `f` is drawn in three strokes. Two dahsed strokes correspond to the positive values of `f`, a dotted stroke corresponds to negative values. Finally, a cubic polynomials `g` is plotted and a table of points is written to the console and log file (figure 3).

```
numeric u;
u := 0.5cm;
  %%% Draw a coordinate system.
```

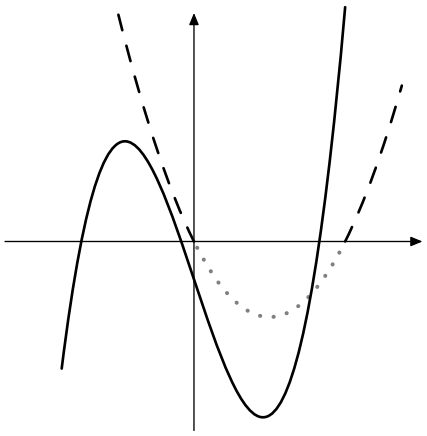

Figure 3: Two polynomials in a coordinate system.

```
  xmin := -5; xmax := 6;
  ymin := -5; ymax := 6;
  drawarrow ((xmin,0)--(xmax,0)) scaled u;
  drawarrow ((0,ymin)--(0,ymax)) scaled u;
  drawoptions(withpen pencircle scaled 1bp);
  %%% Define polynomial f of degree 2.
path f[];
  newBPolynomial.f(0, 0.5, -2, 0);
  f1 := f.getPath(-2, 0);
  f2 := f.getPath(0, 4);
  f3 := f.getPath(4, 5.5);
  draw f1 scaled u dashed evenly scaled 2;
  draw f3 scaled u dashed evenly scaled 2;
  draw f2 scaled u dashed withdots
    withpen pencircle scaled 1.5bp withcolor .5white;
  %%% Define polynomial g of degree 3.
path g;
  newBPolynomial.g(0.3, 0, -3, -1);
  g := g.getPath(-3.5, 4);
  show g;
  draw g scaled u;
  %%% Write table with some points of g to log file.
  show "Polynomial: " & decimal g.a & "x^3+" & decimal g.b
    & "x^2+" & decimal g.c & "x+" & decimal g.d;
  for x=-5 upto 5:
    show (x, g.eval(x));
  endfor
```

Note command `show g` that writes path `g` to the `log` file. Inspecting that we can easily verify, that `g` consists of just one path segment:

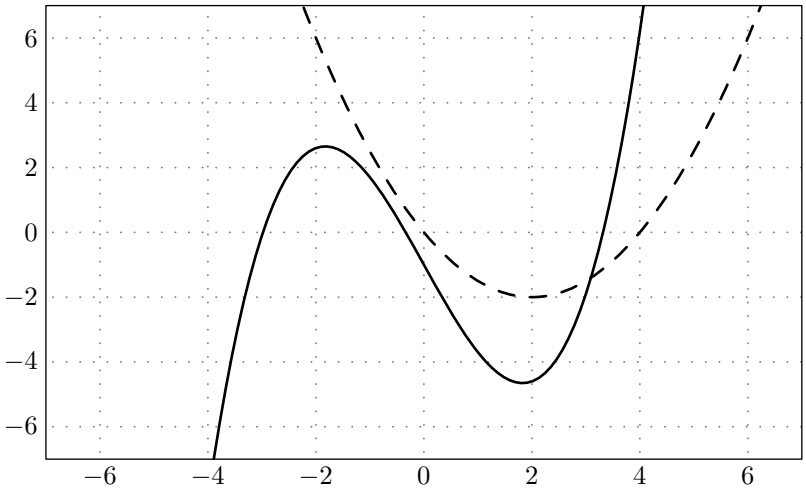

Figure 4: Packages `bpolynomial.mp` and `graph` interacting.

```
(-3.5,-3.36273)..controls (-1,16.70013) and (1.5,-22.30025)..(4,6.2002)
```

The next example demonstrates how `bpolynomial.mp` and John Hobby's `graph` package[1] can be used together to draw polynomials in a coordinate system. Instead of `draw` paths have just to be drawn with a `gdraw` command. The latter macro additionally clips paths to the boundaries of the coordinate system (figure 4).

```
path f,g;
  xmin := -7; xmax := 7;
  ymin := -7; ymax := 7;
  newBPolynomial.f(0, 0.5, -2, 0);
  f := f.getPath(xmin, xmax);
  newBPolynomial.g(0.3, 0, -3, -1);
  g := g.getPath(xmin, xmax);
  draw begingraph(10cm, 6cm);
    setrange(xmin,ymin, xmax,ymax);
    autogrid(grid.bot, grid.lft)
      dashed withdots withpen pencircle scaled .7bp withcolor .5white;
    drawoptions(withpen pencircle scaled 1bp);
    gdraw f dashed evenly scaled 2;
    gdraw g;
    drawoptions();
  endgraph;
```

The code of all examples can also be found in file `examples.mp`.

4 Mathematics

A Bézier curve $P(t)$ with end points $A = (x_A, y_A)$ and $D = (x_D, y_D)$ and control points $B = (x_B, y_B)$ and $C = (x_C, y_C)$ is defined as

$$P(t) = \begin{pmatrix} x \\ y \end{pmatrix}(t) = A+3(B-A)t+3(C-2B+A)t^2+(D-3C+3B-A)t^3, \quad 0 \le t \le 1. \tag{3}$$

An arbitrary function $y = f(x)$ can be written in parameter form as

$$F(t) = \begin{pmatrix} x \\ y \end{pmatrix}(t) = \begin{pmatrix} x(t) \\ f\big(x(t)\big) \end{pmatrix}, \quad t \in \mathbb{R} \tag{4}$$

with parameter t.

For a function

$$f(x) = ax^3 + bx^2 + cx + d, \quad x \in [x_0, x_1] \tag{5}$$

we have

$$x(t) = x_0 + (x_1 - x_0)t, \quad 0 \le t \le 1 \tag{6}$$

and hence

$$F(t) = \begin{pmatrix} x_0 + (x_1 - x_0)t \\ ax(t)^3 + bx(t)^2 + cx(t) + d \end{pmatrix}, \quad 0 \le t \le 1. \tag{7}$$

Writing F(t) down explicitly is left as an exercise for the interested reader.

Finally, setting

$$P(t) = F(t) \tag{8}$$

and sorting the coefficients of the t^k one arrives at the following *original* equation system:

$$x_A = x_0 \tag{9}$$
$$3(x_B - x_A) = x_1 - x_0 \tag{10}$$
$$3(x_C - 2x_B + x_A) = 0 \tag{11}$$
$$x_D - 3x_C + 3x_B - x_A = 0 \tag{12}$$
$$y_A = ax_0^3 + bx_0^2 + cx_0 + d \tag{13}$$
$$3(y_B - y_A) = 3ax_0^2(x_1 - x_0) + 2bx_0(x_1 - x_0) + c(x_1 - x_0) \tag{14}$$
$$3(y_C - 2y_B + y_A) = 3ax_0(x_1 - x_0)^2 + b(x_1 - x_0)^2 \tag{15}$$
$$y_D - 3y_C + 3y_B - y_A = a(x_1 - x_0)^3 \tag{16}$$

Note, there are only constants on the right-hand side of all equations. That is, this equation system is linear in the eight variables x_A, x_B, x_C, x_D, y_A, y_B, y_C, y_D.

Since MetaPost can solve linear equation systems, hacking equations 9 to 16 into MetaPost code and requesting a path segment

(x_A,y_A)`..controls` (x_B,y_B) `and` (x_C,y_C)`..`(x_D,y_D)

returns the polynomial shaped curve we are looking for.

Internally, the `bpolynomial.mp` package does not solve the original equation system, but a *modified* variant, that is numerically slightly more robust.

Equations 9 to 12 can be written down explicitly as

$$x_A = x_0 \tag{17}$$

$$x_B = x_0 + \frac{1}{3}(x_1 - x_0) \tag{18}$$

$$x_C = x_1 - \frac{1}{3}(x_1 - x_0) \tag{19}$$

$$x_D = x_1 \tag{20}$$

Additionally, we know that $D = (x_D, y_D)$ is a point on the polynomial. Therefore, equation 16 of the original system can be replaced by

$$y_D = ax_1^3 + bx_1^2 + cx_1 + d \tag{21}$$

Equations 13 to 15 of the original equation system and the new equations 17 to 21 constitute the modified equation system, that is solved in `bpolynomial.mp`.

Happy TEXing!
Stephan Hennig

References

[1] Hobby, John D., *Drawing graphs with MetaPost*, `http://www.tug.org/docs/metapost/mpgraph.pdf`

[2] Luecking, Dan, *Macros to compute splines*, 2005, `CTAN:graphics/metapost/contrib/macros/splines/splines.pdf`