The bpolynomial package*

Stephan Hennig†

November 28, 2007

Abstract

The MetaPost package bpolynomial 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. Additionally, paths of derivatives and tangents can be calculated.

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. [4] 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 v0.4, last revised 11/28/2007.

†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 `newBPolynomial`

The bpolynomial package provides just one macro `newBPolynomial`. This macro takes one suffix argumant and four numeric arguments 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, numbers 2, 0, −3, −1 match the coefficients of our function f. The suffix argument `f` serves as an identifier where some names of macros and variables, that have to be called later, are derived from. To be more precise, command

```
newBPolynomial.<suffix>
```

defines three new macros

```
<suffix>.getPath
<suffix>.eval
<suffix>.getTangent
```

that do the real work. These macros are described in the following sections.

2.2 `<suffix>.getPath`

Macro `<suffix>.getPath(xmin, xmax)` returns a path exactly matching the polynomial defined by `newBPolynomial.<suffix>` on an 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) transformed T;
```

Note, since the base unit of MetaPost is a big point (1 bp) in most cases polynomials have to be scaled to a proper size before plotting. It is *not* recommended, however, to apply scaling to the polynomial coefficients, since current MetaPost versions[2] can't handle large numbers very well. Instead, scaling should be applied to the path during the `draw` operation. In this manual, scaling is applied by an affine transform

```
T = identity xscaled 10mm yscaled 1mm;
```

[2] At the time of writing the latest release is MetaPost v1.002.

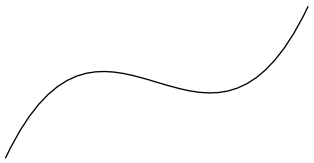

Figure 1: A cubic polynomial.

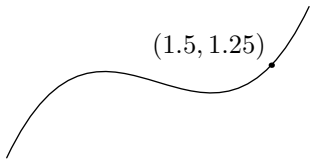

Figure 2: With a labelled point.

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⟩.

Hint: Since the bpolynomial 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.

```
newBPolynomial.f(2, 0, -3, -1);
path f;
f := f.getPath(-2, 2);
draw f transformed T;
```

2.3 `<suffix>.eval`

Macro `<suffix>.eval` can be used to evaluate polynomial ⟨suffix⟩ at a given x-coordinate. The macro takes one argument—the x-coordinate. Labelling an arbitrary point on a polynomial can be done as follows (figure 2).

```
newBPolynomial.f(2, 0, -3, -1);
draw f.getPath(-2, 2) transformed T;
x := 1.5;
show (x, f.eval(x));
dotlabeldiam := 2bp;
dotlabel.ulft(btex $(1.5, 1.25)$ etex, (x, f.eval(x)) transformed T);
```

Note, the label has been provided explicitly in this example (after reading the coordinates off the `log` file). It is also possible to attach the correct coordinates automatically with the help of the MetaPost package LaTeXMP and the LaTeX package numprint. While the former helps passing dynamically generated text from MetaPost to LaTeX, the latter can be used to format and round numbers. [1, 5].

2.4 <suffix>.getTangent

Macro <suffix>.getTangent returns a path tangent to polynomial ⟨suffix⟩. This macro takes three numeric arguments, the coordinate x where the tangent should touch polynomial ⟨suffix⟩, and two values ϵ_-, ϵ_+ that specify the range the tangent is drawn in. These arguments are not the range boundaries, but the neighbourhood around x. The range is $[x + \epsilon_-, x + \epsilon_+]$. This syntax has been choosen to make is easier to move a tangent along a polynomial, keeping the neighbourhood fixed.

As an example, the following code draws a tangent that touches f at $x = -1$ with a neighbourhood $\epsilon = \pm 1$ (figure 3).

```
newBPolynomial.f(2, 0, -3, -1);
draw f.getPath(-2, 2) transformed T;
draw f.getTangent(-1, -1, 1) transformed T;
```

2.5 Dealing with derivatives

Additionally to drawing polynomials, the bpolynomial package supports drawing derivatives of polynomials and tangents thereof. In section 2.1 it was said macro newBPolynomial defines three new macros. But this is not the full story. In fact, the command

newBPolynomial.<suffix>

defines twelve macros, three of them we already know, <suffix>.getPath, <suffix>.eval, <suffix>.getTangent. The remaining nine macros are similar, but correspond to the first, second and third derivative of a polynomial ⟨suffix⟩, resp. To access these macros just add the required number of prime characters to the suffix name (three at maximum). For instance, to get the path corresponding to the first derivative of polynomial ⟨suffix⟩ call

<suffix>'.getPath

and to get a tangent on the first derivative call

<suffix>'.getTangent.

In total these are the macros defined by newBPolynomial.<suffix>:

<suffix>.getPath	<suffix>''.getPath
<suffix>.eval	<suffix>''.eval
<suffix>.getTangent	<suffix>''.getTangent
<suffix>'.getPath	<suffix>'''.getPath
<suffix>'.eval	<suffix>'''.eval
<suffix>'.getTangent	<suffix>'''.getTangent

As an example, the following code draws a tangent on the first derivative of a polynomial f (figure 4).

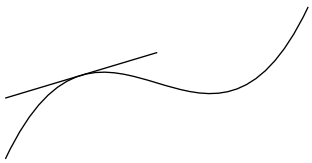

Figure 3: Cubic polynomial with a tangent.

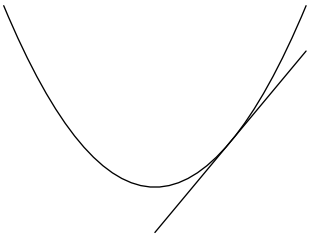

Figure 4: First derivative with a tangent.

```
newBPolynomial.f(2, 0, -3, -1);
draw f'.getPath(-2, 2) transformed T;
draw f'.getTangent(1, -1, 1) transformed T;
```

2.6 Accessing polynomial coefficients

The coefficients passed to `newBPolynomial.<suffix>` are saved in variables `<suffix>.a`, `<suffix>.b`, `<suffix>.c` and `<suffix>.d` and can be accessed by the user.

3 Examples

This section contains some more elaborate examples. The code of all examples can also be found in file `examples.mp`.

In the first example a simple coordinate system is drawn manually. Then a quadratic polynomial `f` is drawn in three strokes. Two dashed strokes correspond to the positive values of `f`, a dotted stroke corresponds to negative values. Finally, a cubic polynomial `g` is plotted and a table of points is written to the console and log file (figure 5).

```
numeric u;
u := 0.5cm;
  %%% Draw 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);
```

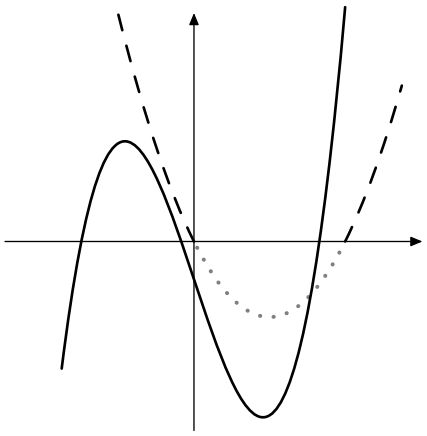

Figure 5: Two polynomials in a coordinate system.

```
  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 `g` consists of just one path segment:

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

The next example demonstrates how bpolynomial and John Hobby's graph package[2] 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 6).

```
path f,g;
```

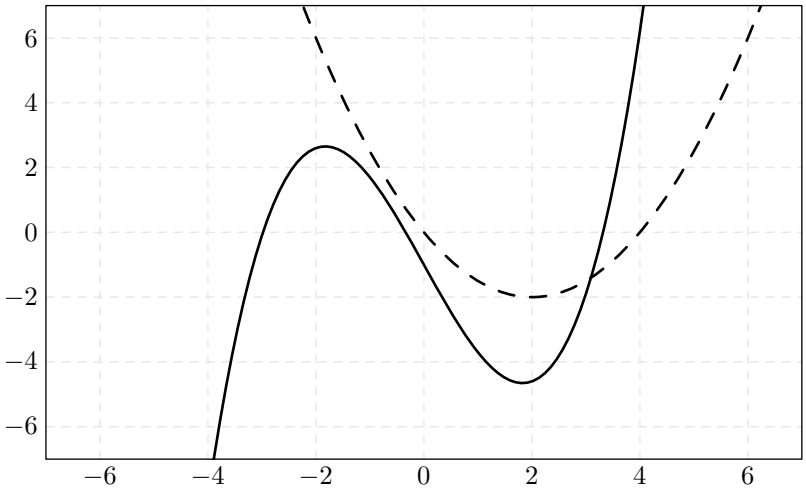

Figure 6: Packages bpolynomial and graph interacting.

```
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 evenly withcolor .9white;
  drawoptions(withpen pencircle scaled 1bp);
  gdraw f dashed evenly scaled 2;
  gdraw g;
  drawoptions();
endgraph;
```

In the last example a cubic polynomial f is drawn together with its derivatives f', f'' and f'''. Additionally, for all four functions the tangents are drawn at $x = 2$. Admittedly, the plot is a little bit crowded. But it should only serve as an example (figure 7).

```
xmin := -6; xmax := 6;
ymin := -6; ymax := 6;
newBPolynomial.f(0.3, -0.5, -0.5, -1);
draw begingraph(10cm, 6cm);
  setrange(xmin,ymin, xmax,ymax);
  autogrid(grid.bot, grid.lft) dashed evenly withcolor .9white;
  drawoptions(withpen pencircle scaled 1bp);
  %%% Draw f and its derivatives f', f'', f'''.
```

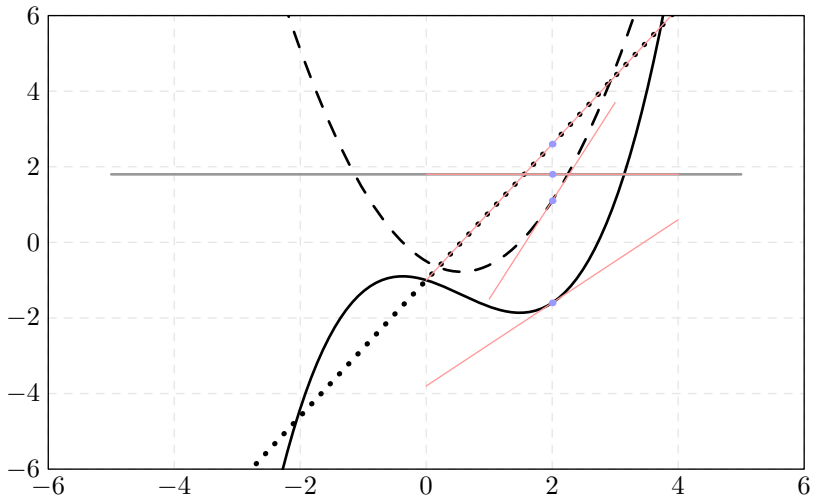

Figure 7: A cubic polynomial with derivatives and tangents.

```
  gdraw f.getPath(xmin, xmax);
  gdraw f'.getPath(xmin, xmax) dashed evenly scaled 2;
  gdraw f''.getPath(xmin, xmax) dashed withdots
    withpen pencircle scaled 2bp;
  gdraw f'''.getPath(-5, 5) withcolor .6white;
  %%% Draw tangents and mark points.
  x := 2;
  drawoptions(withcolor red+.6(green+blue));
  gdraw f.getTangent(x, -2, 2);
  gdraw f'.getTangent(x, -1, 1);
  gdraw f''.getTangent(x, -2, 2);
  gdraw f'''.getTangent(x, -2, 2);
  drawoptions(withcolor blue+.6(red+green));
  dotlabeldiam := 2.5bp;
  gdotlabel("", (x, f.eval(x)));
  gdotlabel("", (x, f'.eval(x)));
  gdotlabel("", (x, f''.eval(x)));
  gdotlabel("", (x, f'''.eval(x)));
  drawoptions();
endgraph;
```

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 [3]

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

This equation can be rewritten as

$$P(t) = A + 3(B-A)t + 3(C-2B+A)t^2 + (D-3C+3B-A)t^3, \quad 0 \le t \le 1. \tag{4}$$

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

$$F(t) = \begin{pmatrix} x \\ y \end{pmatrix}(t) = \begin{pmatrix} x(t) \\ f\big(x(t)\big) \end{pmatrix}, \quad t \in \mathbb{R} \tag{5}$$

with parameter t.

For a polynomial function

$$f(x) = ax^3 + bx^2 + cx + d, \quad x \in [x_0, x_1] \tag{6}$$

we have

$$x(t) = x_0 + (x_1 - x_0)t, \quad 0 \le t \le 1 \tag{7}$$

and hence

$$F(t) = \begin{pmatrix} x_0 + (x_1 - x_0)t \\ ax(t)^3 + bx(t)^2 + cx(t) + d \end{pmatrix}, \quad 0 \le t \le 1. \tag{8}$$

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

Finally, setting

$$P(t) = F(t) \tag{9}$$

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

$$x_A = x_0 \tag{10}$$
$$3(x_B - x_A) = x_1 - x_0 \tag{11}$$
$$3(x_C - 2x_B + x_A) = 0 \tag{12}$$
$$x_D - 3x_C + 3x_B - x_A = 0 \tag{13}$$
$$y_A = ax_0^3 + bx_0^2 + cx_0 + d \tag{14}$$
$$3(y_B - y_A) = 3ax_0^2(x_1 - x_0) + 2bx_0(x_1 - x_0) + c(x_1 - x_0) \tag{15}$$
$$3(y_C - 2y_B + y_A) = 3ax_0(x_1 - x_0)^2 + b(x_1 - x_0)^2 \tag{16}$$
$$y_D - 3y_C + 3y_B - y_A = a(x_1 - x_0)^3 \tag{17}$$

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 10 to 17 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 package does not solve the original equation system, but a *modified* variant, that is numerically slightly more robust.

Equations 10 to 13 can be written down explicitly as

$$x_A = x_0 \tag{18}$$

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

$$x_C = x_1 - \frac{1}{3}(x_1 - x_0) \tag{20}$$

$$x_D = x_1 \tag{21}$$

Additionally, we know that $D = (x_D, y_D)$ is a point on the polynomial. Therefore, equation 17 of the original system can be replaced by

$$y_D = ax_1^3 + bx_1^2 + cx_1 + d \tag{22}$$

Equations 14 to 16 of the original equation system and the new equations 18 to 22 constitute the modified equation system, that is solved in bpolynomial.

Happy TEXing!
Stephan Hennig

References

[1] Harders, Harald, *The numprint package*, 2007, CTAN:macros/latex/contrib/numprint/numprint.pdf

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

[3] Knuth, Donald E., *The METAFONTbook*, Addison-Wesley, Reading, Massachusetts, 1986, (Computers & Typesetting, C)

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

[5] Morawski, Jens-Uwe, *latexMP*, 2005, CTAN: graphics/metapost/contrib/macros/latexmp/doc/latexmp.pdf