Evaluating polynomials

Being able to graph arbitrary polynomials is very useful. In order to do this efficiently, a good way to evaluate the polynomial is required, which of course means that we should design a PostScript procedure to evaluate an arbitrary polynomial $P(x)$ for an arbitrary value of x. In other words, this procedure will have two arguments, the first an array $[a_0\ a_1\ \ldots\ a_{n-1}]$ to be interpreted as the coefficients of the polynomial $P(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$, and the other the value of x.

1. The most straightforward way to do it

Here, for example, is a procedure which will evaluate an arbitrary polynomial of degree three.

```
/cubic-poly {
2 dict begin
/x exch def
/a exch def
a 0 get
a 1 get x mul add
a 2 get x 2 exp mul add
a 3 get x 3 exp mul add
end
} def
```

Exercise 1.1. *Extend this procedure, using a* **for** *loop, so that it will evaluate a polynomial of arbitrary degree. Be careful that your procedure works even for a polynomial of degree* 0 *(a constant). Also, it should return* 0 *if the array is empty. (Note that the degree is one less than the length of the coefficient array.)*

Exercise 1.2. *Extend in turn the procedure from the previous exercise so that it will return an array of two numbers* $[P(x)\ P'(x)]$.

2. Horner's method

The PostScript command **exp** is somewhat slow, and the straightforward procedure used above is therefore probably inefficient. Better is an elegant method of evaluating polynomials due to the nineteenth century English mathematician Horner. It does not use **exp**, but gets by with just successive multiplications and additions.

We start off by rewriting a cubic polynomial:

$$P(x) = a_3x^3 + a_2x^2 + a_1x + a_0 = (((a_3)x + a_2)x + a_1)x + a_0\ .$$

In other words, we evaluate in succession the numbers

$$\begin{aligned} &a_3 \\ &a_3x + a_2 \\ &(a_3x + a_2)x + a_1 \\ &(((a_3)x + a_2)x + a_1)x + a_0 \end{aligned}$$

or, in other words, giving these expressions labels, we calculate

$$\begin{aligned} b_2 &= a_3 \\ b_1 &= b_2x + a_2 \\ b_0 &= b_1x + a_1 \\ b_{-1} &= b_0x + a_0\ . \end{aligned}$$

At the end $b_{-1} = P(x)$. In PostScript this becomes

```
/b a 3 get def
/b b x mul a 2 get add def
/b b x mul a 1 get add def
/b b x mul a 0 get add def
```

and at the end $b = P(x)$. We can even get by without definitions.

```
a 3 get
x mul a 2 get add
x mul a 1 get add
x mul a 0 get add
```

will leave $P(x)$ on the stack. There is the germ of a simple loop here.

Exercise 2.1. *Write a procedure which evaluates an arbitrary fourth degree polynomial $P(x)$. Do two versions, one straightforward as above, the other using a loop, and without defining b.*

Exercise 2.2. *Write a PostScript procedure which extends this to evaluate an abitrary $P(x)$ by Horner's method.*

3. Evaluating the derivatives efficiently

Very often in plotting a graph it is useful to obtain the value of $P'(x)$ at the same time as $P(x)$. Horner's method allows this to be done with little extra work.

The formulas for the b_i above can be rewritten as

$$\begin{aligned} a_3 &= b_2 \\ a_2 &= b_1 - b_2 x \\ a_1 &= b_0 - b_1 x \\ a_0 &= b_{-1} - b_0 x \end{aligned}$$

We can therefore write

$$\begin{aligned} P(X) &= a_3 X^3 + a_2 X^2 + a_1 X + a_0 \\ &= b_2 X^3 + (b_1 - b_2 x) X^2 + (b_0 - b_1 x) X + (b_{-1} - b_0 x) \\ &= b_2 (X^3 - X^2 x) + b_1 (X^2 - X x) + b_0 (X - x) \\ &= (b_2 X^2 + b_1 X + b_0)(X - x) + b_{-1} \\ &= (b_2 X^2 + b_1 X + b_0)(X - x) + P(x) \ . \end{aligned}$$

Therefore the coefficients b_i have significance in themselves; they are the coefficients of a simple polynomial:

$$b_2 X^2 + b_1 X + b_0 = \frac{P(X) - P(x)}{X - x} = \text{(say) } P_1(X) \ .$$

One remarkable consequence of this is that we can evaluate $P'(x)$ easily since a simple limit argument gives

$$P'(x) = b_2 x^2 + b_1 x + b_0$$

which means that we can apply Horner's method to the polynomial

$$P_1(X) = b_2 X^2 + b_1 X + b_0$$

in turn to find it. We can in fact evaluate $P(x)$ and $P'(x)$ more or less simultaneously, if we think a bit about it. Let the numbers c_i be calculated from $P_1(X)$ in the same way that the b's came from $P(X) = P_0$.

$$\begin{aligned} b_2 &= a_3 \\ b_1 &= b_2 x + a_2 \\ b_0 &= b_1 x + a_1 \\ b_{-1} &= b_0 x + a_0 \\ c_1 &= b_2 \\ c_0 &= c_1 x + b_1 \\ c_{-1} &= c_0 x + b_0 \end{aligned}$$

concluding with $P(x) = b_{-1}$, $P'(x) = c_{-1}$. This requires that we store the values of b_i to be used in calculating the c's. In fact we can avoid this by interlacing the calculations:

$$\begin{aligned} b_2 &= a_3 \\ c_1 &= b_2 \\ b_1 &= b_2 x + a_2 \\ c_0 &= c_1 x + b_1 \\ b_0 &= b_1 x + a_1 \\ c_{-1} &= c_0 x + b_0 \\ b_{-1} &= b_0 x + a_0 \end{aligned}$$

concluding with $P(x) = b_{-1}$, $P'(x) = c_{-1}$.

Exercise 3.1. *Design a procedure with two arguments, an array and a number x, which returns $[P(x)\ P'(x)]$, where the polynomial P is defined by the array argument. It should use Horner's method to do this, preferably in the most efficient version. (Hint: evaluatng the derivative requires one less step than evaluating the polynomial. This can be dealt with by initializing c to 0 in the steps shown above. In this way, c and b can be handled in the same number of steps, and writing the loop becomes simpler.)*