Matlab Notes, Part V

Matlab and Polynomials

The Taylor series (centered at \(a\)) for a given function \(f(x)\) is
\[
c_0 + c_1(x - a) + c_2(x - a)^2 + \ldots + c_n(x - a)^n + c_{n+1}(x - a)^{n+1} + \ldots
\]
where the \(n^{th}\) Taylor coefficient \(c_n = \frac{f^{(n)}(a)}{n!}\).

The partial sum of the series out to the \(x^n\) term is called the \(n^{th}\) degree Taylor polynomial for \(f(x)\) at \(a\):
\[
T_n(x) = c_0 + c_2(x - a) + c_2(x - a)^2 + \ldots + c_n(x - a)^n
\]
For a given \(x\), saying “the sum of the Taylor series = \(f(x)\)” means (by definition of “sum” of a series) that
\[
\lim_{n \to \infty} T_n(x) = f(x)
\]
In other words, if \(f(x) = \text{sum of its Taylor series at } a\), then the Taylor polynomials \(T_n(x)\) better and better approximate \(f(x)\) as \(n \to \infty\).

We want to plot some Taylor polynomials to see what all this looks like. Therefore it’s convenient to know a little more about some special tricks in Matlab for handling polynomials.

For a “short” polynomial (“low degree”) we can simply define the polynomial in a function m-file and proceed to plot and calculate. However, for larger values of \(n\) (suppose the polynomial had degree 20 – or even degree 100) typing in the whole formula definition the polynomial might be rather tedious (and prone to typing errors, as well). So we’d like a shortcut.

To describe a polynomial, all you need to know is the coefficient of each power of \(x\) – that is, the array of coefficients contains all the necessary information. For example, the polynomial
\[
P(x) = x^4 + 3x^2 - 2x + 1
\]
could be represented in Matlab by the array of coefficients
\[
p = [1,0,3,-2,1].
\]
Matlab assumes that the coefficients (left to right) correspond to decreasing powers of \(x\);

“0” coefficients must be included in the array for missing powers of \(x\) – such as the 3\(^{rd}\) degree term in the example.

If a polynomial has a high degree, it’s much easier just to represent it as an array of coefficients than to type in the whole formula with all the powers of \(x\) included.

Matlab recognizes that the array of numbers “\(p\)” is supposed to represent a polynomial whenever “\(p\)” appears in certain special commands. One of these is “polyval” (for “polynomial value”). If “\(a\)” is a number, then polyval(p,a) gives the value of the polynomial with coefficients \(p\) when \(x = a\).

For the polynomial array “\(p\)” above, check that you understand what each of these commands is computing:

polyval(p,0) gives 1
polyval(p,1) gives 3
polyval(p, -2) gives 33
To graph the polynomial with coefficients in the array `p` over the interval \([-2,2]\), you could use, for example,

\[
x = \text{linspace}(-2, 2, 500);
y = \text{polyval}(p, x);
\text{plot}(x, y)
\]

A minor inconvenience: the standard mathematical way of writing Taylor polynomials is in order of increasing powers of `x`: for example, perhaps \(T_3(x) = 2 + x + 3x^2 + x^3\). Therefore, to represent \(T_3(x)\) in Matlab, you would reverse the order to descending powers and enter an array – named `t3`, perhaps – as \([1,3,1,2]\).

**Reversing the order of elements in an array**

It may happen that the coefficients in an array are in “reverse” order from what you want – for example, a long array of polynomial coefficients may have been generated by a loop which produced the coefficients of terms with powers of `x` in ascending order. Rather than retyping the array to reverse the get the coefficients into “descending order”, we can do something like the following:

Suppose we have an array `c = [2,1,3,1]` and we want to reverse the order to get the Matlab array (“descending power order”) representing the polynomial \(T_3\) above. To do this, use the command

\[
t3 = c(4: -1: 1)
\]

This use of array addressing means: “to get the elements in `t3`, start with the 4\textsuperscript{th} element of “a” and list the elements of “a” down to the 1\textsuperscript{st} element, proceeding with steps of size – 1”

This produces \(t3 = [c(4), c(3), c(2), c(1)]\), that is, \(t3 = [1,3,1,2]\)

If “a” has a large number of elements (or if you’ve forgotten exactly how many) you can simply let Matlab count the element in “a” for you. You can then make a new array with the elements in reversed order with the command

\[
c(\text{length}(c): -1: 1)
\]