1 Introduction to Polynomials

Unsurprisingly, polynomials are useful classes of functions, both analytically and computationally. You might think that using an n^{th} order polynomial for something is a sufficient description. Analytically, you would be (more-or-less) correct, however nothing could be further from the truth computationally. The main problem is that, at least as traditionally written, we can approximate polynomials using other polynomials with increasingly high accuracy as the order increases. For instance, x^{20} looks an awful lot like $\frac{x^{19}+x^{21}}{2}$ on the interval $[0, 1]$. If you try to describe a function, the computer will in essence throw up its hands and say "I can't tell the difference", which usually leads to huge roundoff errors. If working in single precision, this frequently happens with polynomials whose orders are in the upper single digits. If you aren't careful with scaling x-ranges, it can happen even sooner.

The solution is to use different classes of polynomials that have better numerical properties. Usually these classes are defined via a *recurrence relation* (the $n+1^{th}$ polynomial is defined as a sum of some number of previous polynomials), and have some orthogonality property.

2 Standard Polynomials

Setting the stage, we can cast our usual polynomials in the language we will use to describe other classes. If we have

$$y = \sum_{n=0}^{N} c_n x^n$$

then we can think of the set of x^n as basis functions, and the coefficients c_n tell us how much of each basis function our final function y contains. Further, let us define $q_n \equiv x^n$, so the q_n are our basis functions. Of course, we could simply write down each q_n, but note that we could equally well use a recurrence relation:

$$q_{n+1} = xq_n, q_0 = 1$$

We start with q_0, and every time we increase the index, we multiply by a factor of x, so $q_1 = x$, $q_2 = x^2$ and so on. It doesn't make a much of a difference for standard polynomials, but these recurrence relations usually form a much more numerically stable way of evaluating other classes of polynomials.

Finally, note that we could write our expression for y as a matrix equation. If we want to evaluate y at a discrete set of x_i, then let Q be the matrix whose columns are the q_n evaluated at each x point, and let c be the vector made up of the c_n. Then we have:

$$y = \mathrm{Q}c$$

Writing things this way will allow us to invoke the considerable power of linear algebra when working with polynomials (or indeed, any set of basis functions we can write as a matrix multiplication).

3 Chebyshev Polynomials

One broadly useful class of polynomials are the Chebyshevs T_n. We will start with them because their recurrence relation is easy to derive, and many of their useful properties can be inferred just from their definition.

The fundamental definition of a Chebyshev polynomial[1] is:

$$T_n = \cos(n \arccos(x))$$

defined on $-1 \leq x \leq 1$. It isn't obvious that these would be polynomials, but we can show that they are. To do this, we'll expand T_{n+1} and T_{n-1}.

$$T_{n\pm1} = \cos((n\pm1)\arccos(x) = \cos(n\arccos(x))\cos(\arccos(x)) \mp \sin(n\arccos(x))\sin(\arccos(x))$$

If we add them together, the sin terms will cancel and we are left with

$$T_{n+1} + T_{n-1} = 2\cos(n\arccos(x))\cos(\arccos(x))$$

We know that $\cos(n\arccos(x))$ is just T_n, and $\cos(\arccos(x))$ is just x, so we have

$$T_{n+1} + T_{n-1} = 2xT_n$$

or

$$T_{n+1} = 2xT_n - T_{n-1}$$

From the funamental definition, we can read off the first two polynomials that can get us started: $T_0 = \cos(0\arccos(x)) = 1, T_1 = \cos(\arccos(x)) = x$. If T_n is an n^{th} order polynomial, then because T_{n+1} picks up an xT_n in it, then T_{n+1} will be an $n+1^{th}$ order polynomial. We can use the recurrence relation to write down a few extra polynomials: $T_2 = 2xT_1 - T_0 = 2x^2 - 1$, $T_3 = 2xT_2 - T_1 = 4x^3 - 2x - x = 4x^3 - 3x$, etc.

Let's look at some properties of Chebyshev polynomials. Because the output of cos is bounded to be between -1 and 1, then the values of all of the Chebyshev's are bounded between -1 and 1. Furthermore, because the output of arccos is between 0 and π, the input to cos of the n^{th} polynomial will go from 0 to $n\pi$, and T_n will hit ±1 exactly $n+1$ times for $-1 \leq x \leq 1$. Basically, each T_n has n half-periods of a cosine, but with a bit of horizontal (but not vertical) distortion. The first few T_n are plotted in Figure 1. Less obviously, the T_n are orthogonal under a suitably-chosen weight:

$$\int_{x=-1}^{1} T_n T_m \frac{dx}{\sqrt{1-x^2}} \propto \delta_{m,n}$$

The fact that the T_n are bounded by ±1 makes them extremely useful in some applications of function approximation. Normally, linear algebra will give us the *least-squares* solution to a problem, which minimizes the total error squared.

[1] Technically, these are Chebyshev polynomials of the first kind. There are also Chebyshev polynomials of the second kind, but we will not be addressing them here.

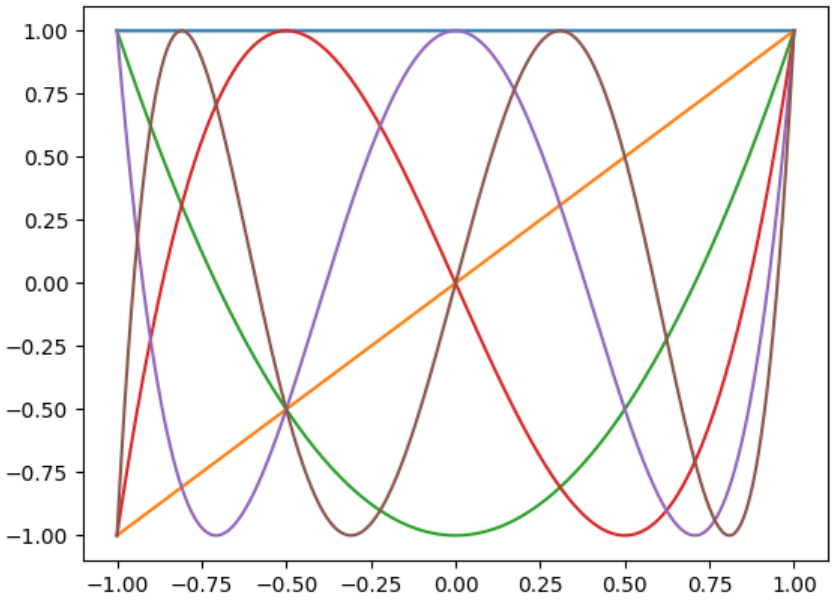

Figure 1: The first 6 Chebyshev polynomials. Note that their zeroes are (roughly) equally spaced, and the minima/maxima are all ± 1.

Usually, this means that our function will be extremely well modelled in the center of our domain, with errors rising sharply at the edges. This is fine for a lot of cases, but let's say we wanted a polynomial approximation to some transcendental function (say $\exp, \sin, \cos, \log$...). Usually, we'd like our error to be small everywhere, not extra-small in some regions and not so great in other regions. Chebyshevs provide a nice way of doing this. Let's say we had some high-order Chebyshev fit to our function:

$$y = \sum_{n=0}^{N} T_n c_n$$

but then decide to only keep the first M terms. The maximum error we can introduce by this truncation is

$$\sum_{n=M+1}^{N} |c_n|$$

Usually, the c_n drop pretty quickly with n, and so we can read off the maximum truncation error by keeping just the first m terms by summing the absolute

values of the coefficients of the terms we drop. Since the minima/maxima are roughly evenly-spaced, this also means the truncation error tends to be spread somewhat uniformly across the interval, as opposed to bunched up at the edges.

We can look at the properties of the Chebyshev's we already have to highlight some important numerical traits as well. First, note that the coefficients of a Chebyshev written out will always be integers, because the $n+1^{th}$ polynomial doubles the coefficients of the n^{th} polynomial, shifts them one higher (because of the multiplication by x), and adds in the coefficients to the $n-1^{th}$ polynomial. Because $T_0 = 1$ and $T_1 = x$ have integer coefficients, every T_n will have integer coefficients. Second, the coefficient of x^n in T_n must be 2^{n-1}, since we always double the leading order term in the recurrence relation, and the leading coefficient of T_1 is 1. Finally, the sum of coefficients must be ± 1 because $T_n(1) = \pm 1$ from the fundamental definition, and the fact that a polynomial evaluated at $x = 1$ is just the sum of the coefficients. Combined, these facts mean that the coefficients of the Chebyshev polynomials, when written out as traditional polynomials, are exploding exponentially, but delicately cancelling out so the output stays bounded on $[-1, 1]$. This is a hallmark of cases where you can have catastrophic precision loss since you are differencing huge numbers to get something small. Always, always use the recurrence relations to evaluate your Chebyshev polynomials and resist the temptation to write out the coefficients.

One more important fact we can address is the *stability* of the recurrence relation. If you make a small error (which you inevitably will do when working with finite-precision computers), does the induced error grow with time? If yes, your relation may be technically accurate but in practice might be useless, because exponentially growing roundoff errors quickly lead to garbage. Fortunately, we can pretty much carry out a stability analysis of the T_n by inspection. Note that n does not appear anywhere in the recurrence relation, so if you add a little bit of error somewhere along your generation of the $T_n's$, it looks like you've started a new copy of the recurrence relation, scaled by the error you made. Since this new copy stays bounded by $[-1, 1]$ times your initial error, any error you make will neither grow nor shrink with time. The recurrence relation is *marginally stable*. For computing, marginally stable is fine - we always will have roundoff error, and since any roundoff error stays at the level of the roundoff error, life will be good. The more common case is that the relation is either stable or unstable - if stable, we can use it as-is. Unstable relations can also be useful, because if it something is unstable as we move forward, it is in general (but not always) stable if we move backwards[2].

[2]Bessel functions are a classic case of this - the relation is unstable in the forward direction, but stable in the backwards. You start off at some order a bit higher than you care about with any old starting values, and run backwards. Very quickly, you converge to the right answer, modulo an overall scaling. If you know the zero-order Bessel function, you can rescale your values by that to get the true higher-order Bessel functions.

4 Legendre Polynomials

Legendre polynomials P_n are perhaps the most widely-used class of polynomials in computing/physics (the fact that they got P as their variable is a bit of a tip-off). They appear naturally in separation-of-variables solutions in spherical coordinates, and are explicitly orthogonal to each other on $[-1, 1]$:

$$\int_{x=-1}^{1} P_n(x)P_m(x)dx = \frac{2}{2n+1}\delta_{m,n}$$

The recurrence relation is:

$$(n+1)P_{n+1} = (2n+1)xP_n - nP_{n-1}$$

with the starting conditions $P_0 = 1, P_1 = x$ (just like Chebyshevs). The presence of n in the recurrence relation makes stability analysis a bit trickier, but it turns out that this relation is also stable. The first few Legendre polynomials are plotted in Figure 2.

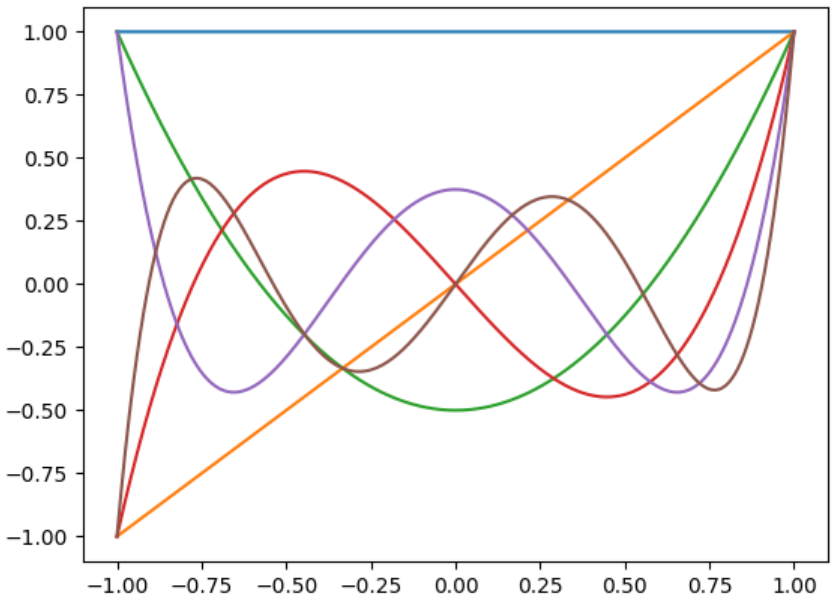

Figure 2: The first 6 Legendre polynomials. Note that their zeroes are (roughly) equally spaced, and they go to ± 1 on the edges, but unlike for Chebyshevs, the minima/maxima are less than one in the interior.

Legendre polynomials show up in physics (and other fields) because the set of $P_n \cos(\theta)$ are orthogonal on the surface of the sphere, so any spherical problem

with no dependence on ϕ will have its solution in terms of the P_n. They arise particularly often in quantum and E&M. Their orthogonality conditions means they also are particularly useful in numerical integration. Consider a function $f(x) = \sum P_n(x)c_n$. What then is $\int_{-1}^{1} f(x)\mathrm{d}x$?

$$\int_{-1}^{1} f(x)\mathrm{d}x = \int \sum (P_n c_n)\mathrm{d}x = \int \sum (P_n c_n)1\mathrm{d}x = \int \sum (P_n c_n)P_0\mathrm{d}x$$

We can pull the constant coefficients and the sum out of the integral to get:

$$\sum c_n \int_{-1}^{1} (P_n P_0)\mathrm{d}x = 2c_0$$

where we used the orthogonality condition to see that only the P_0 term contributes to the total area. Another way of phrasing this is that the average value of a function described by Legendre polynomials is just the P_0 coefficient.

We can use the fact that we just need the P_0 coefficient to get the area to pull off a neat trick when numerically integrating functions. Let's say we have N data points, evenly-spaced in x. Let's take the first N Legendre polynomials, then we have $y = Pc$ with P square. It turns out that P is also well-behaved numerically (because the columns are "close to" orthogonal, although they don't quite get there for a finite number of x points), so we can solve for c just by taking the inverse:

$$c = P^{-1}y$$

We only care about c_0, which we get by dotting the first row of P^{-1} against y. If we think of that row as a set of weights, then we know that the area under the Legendre polynomial approximation of y is just the weighted sum of the y values times those weights. We have to be ever so slightly careful about the overall normalization - c_0 gives us the *average* value of y, but we want the total area. The x-range covered by N points is $(N-1)\delta x$ where δx is the spacing bewteen points. If we let w_i be the entries in the first row of P^{-1}, then the total area will be $\sum (N-1)w_i y_i \delta x$. We can generate these coefficients with just a few lines of python:

```
import numpy as np
order=2
N=order+1
x=np.linspace(-1,1,N)
mat=np.polynomial.legendre.legvander(x,order)
mat_inv=np.linalg.inv(mat)
coeffs=mat_inv[0,:]*order
```

If we have order=2, we recover Simpson's rule, but we can just as easily go to higher order by increasing the order parameter.

5 Other Polynomials

While we won't go over them in any detail, there are some other classes of polynomials that can be useful, depending on your situation.

Associated Legendre Polynomials: The associated Legendre "polynomials" $P_{l,m}$ are used to get the fully general set of orthogonal basis functions on the surface of the sphere. Up to normalization coefficients, the orthogonal functions are

$$Y_{l,m} = P_{l,m}(\cos(\theta)) \exp(im\phi)$$

for l, m integers, and $-l \leq m \leq l$. I put polynomials in quotes because the $P_{l,m}$ pick up a factor of $\sqrt{1-x^2}$ for m odd and so aren't actually polynomials at all. They are, however, generated using recurrence relations similar to the ones we have already seen. The angular part of the hydrogen wave function is just the $Y_{l,m}$ times coefficients, where l is the energy quantum number and m is the angular momentum. You've almost certainly seen the first few plotted in a chemistry or quantum textbook, possibly without realizing what you were seeing.

Zernicke Polynomials: The Zernicke polynomials $Z_n^l$are orthogonal on the unit circle. This makes them extremely useful in optics when you have a circular aperture, where wavefront distortions are expressed in terms of Zernicke polynomials. The Z_2^0 term is the defocus (if you wear glasses, the diopter of your prescription is undoing this term), while the $Z_2^{\pm 2}$ terms are astigmatism. If your eyes and/or telescope/binoculars are not great, then the $Z_3^{\pm 1}$ terms appear as coma.

6 Rational Functions

Another closely related and useful class of functions are rational functions, the ratio of two polynomials. Without (much) loss of generality[3], we can write a rational function as:

$$y = \frac{\sum_{i=0}^{M} a_i x^i}{1 + \sum_{j=1}^{N} b_j x^j}$$

Rational functions do a *much* better job describing functions with poles in them, *e.g.* a Lorentzian *is* a rational function, but the poles at $\pm i$ wreak all sorts of havoc with polynomial approximations. Rational functions also (usually) do a much better job extrapolating a function. If the numerator and denominator are the same order, the limiting behavior of a rational function as x approaches $\pm\infty$ is to go to a constant, while a polynomial will blow up. In particular, the rate at which the polynomial blows up is set by the order of the fit, so you basically can't increase the accuracy of your fit in the region where you have

[3]The exception is if you have a pole at $x = 0$. In that case you can put the 1 in the numerator and everything carries through similarly. You can't have the constant term be zero for both the numerator and denominator, because if you did, you could just keep dividing top and bottom through by x until at least one of the constant terms is non-zero.

data without making extrapolation worse. If you know the limiting behavior of your function, you can pick the order of the numerator and denominator to match it. Rational function extrapolation may or may not be *accurate*, but at least you can set the limiting behavior and it won't blow up on you (poles excepted).

Rational functions are nonlinear, but in a way that makes them easy to fit. We can write our model down as

$$y = \frac{Pa}{1 + Qb}$$

where P and Q are matrices of polynomials (possibly of different order), and Q is missing the first column. We can re-write this as

$$y + YQb = Pa$$

$$y = -YQb + Pa$$

where Y is the diagonal matrix of y values, and a and b are the polynomial coefficients for the numerator and denominator. If we stack a and b together, we can also stack P and $-YQ$ together to get a linear equation. If the combined number of terms in a and b is equal to the numnber of points in y, we can solve this with a simple matrix inverse to get the unique rational function of your chosen order that goes through the y points. Of course, you'd better not have $y = 0$ happen... (why?) We'll see later in the course (subject to time constraints) how this can be used as a starting point for some other, possibly more robust ways of using rational functions.