Functions in real life are typically very complicated and messy. One way to handle this is by approximating them using functions that are easier to understand. In fact, the basis of integration theory is the fact that integrable functions can be written as a limit of particularly simple functions (step functions). However, this is a bad idea for numerical computations since the step functions appearing here will be arbitrarily complicated, so impossible to represent on a computer.

Popular choices include polynomials (this unit) and sine/cosine (Fourier analysis). The reason that polynomials are such a common choice is both ease of use (polynomials are very easy to do calculus with) as well as *Weierstrass's theorem*.

Theorem 0.1 (Weierstrass)**.** *If f is continuous on the interval $[a, b]$ then for each $\varepsilon > 0$ there is a polynomial P such that $|f(x) - P(x)| < \varepsilon$ for all $x \in [a, b]$.*

One way we've encountered to approximate a function with polynomials is Taylor polynomials. However, Taylor polynomials are (typically) only accurate locally near a chosen point and quickly become inaccurate even slightly further away. This makes sense: Taylor polynomials only use information about a function at a particular point, so there's generally no reason to expect that the approximation will be valid anywhere else.

1 Interpolation

Instead of only using information about a function at one point, the *interpolation* problem asks for a polynomial that agrees with given function values at specified points. More precisely, given $n + 1$ points $\{(x_i, f(x_i))\}$, find a polynomial P of minimal degree (at most n) such that $P(x_i) = f(x_i)$ for all i. You've already encountered this for degrees 0 and 1.

Example 1.1. If we're only given one point $(x_0, f(x_0))$, then the function $P(x) = f(x_0)$ is the unique degree 0 approximation.

If we're given two points $(x_0, f(x_0))$ and $(x_1, f(x_1))$, then the linear function

$$P(x) = \frac{f(x_1) - f(x_0)}{x_1 - x_0} x + f(x_0) - x_0 \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

is the unique degree 1 approximation (if $x_1 = x_0$, then we don't have a function.)

The degree 2 situation might be less familiar.

Question 1.2. Given three points $(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$, find a degree (at most) 2 polynomial $P(x)$ satisfying $P(x_i) = f(x_i)$.

We need a more principled way to make this work. The first observation is that this can be transformed into a linear algebra problem:

$$\begin{pmatrix} 1 & x_0 & \dots & x_0^n \\ \vdots & & & \vdots \\ 1 & x_n & \dots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f(x_0) \\ \vdots \\ f(x_n) \end{pmatrix}$$

The left matrix is known as a *Vandermonde matrix* and is invertible as long as all the x_i are distinct. Its determinant can be computed by a straightforward application of row-reduction. So if the x_i are distinct, then there is a unique polynomial interpolating the given points. There are various ways to solve this system of equations (and we'll discuss numerical linear algebra later), but for now the important fact we've learned is that there is a *unique* interpolating polynomial, so any polynomial we find will do the job.

1.1 Lagrange

Lagrange (and probably many others before Lagrange...) came up with the following clever polynomials. Suppose we are given $n+1$ points $(x_i, f(x_i))$ for $0 \leq i \leq n$. Then define the polynomials

$$L_{n,k}(x) = \frac{(x-x_0)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_n)}{(x_k-x_0)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_n)}.$$

Question 1.3. What is $L_{n,k}(x_i)$?

How can you use the $L_{n,k}$ to construct a degree n polynomial interpolating the given points?

What are some downsides of this approach? One in particular is that even if we compute the degree n approximation, it's unclear how to use this to compute the degree $n+1$ approximation.

1.2 Newton

An alternative approach to Lagrange interpolation would be to revisit the linear algebra problem we found above. We can get this by triangularizing the Vandermonde matrix, but I'll spare you the details. The result of this is that we can choose to write the polynomial P using the building blocks

$$N_k(x) = (x-x_0)\dots(x-x_{k-1}).$$

Question 1.4. What is $N_k(x_i)$ for $i < k$?

How can we use the N_k to construct a degree n polynomial interpolating the given points? This is a bit trickier than Lagrange, but we can use an inductive formulation.

If we've already found the degree $n-1$ polynomial approximation P_{n-1} using the first n points, then we know that we want

$$P_n(x) = P_{n-1}(x) + a_n N_n(x)$$

Since we want to match the point x_n, this gives

$$f(x_n) = P_n(x_n) = P_{n-1}(x_n) + a_n N_n(x_n),$$

which we rearrange to

$$a_n = \frac{f(x_n) - P_{n-1}(x_n)}{N_n(x_n)}.$$

Question 1.5. What could go wrong with this formula?

If P_{n-1} is already a decent approximation, then it will be close to $f(x_n)$, making the numerator small. We've already seen how this can go wrong in floating point arithmetic. To make things worse, if we distribute the points $x_0, \ldots, x_n$ cleverly, the denominator may be small as well, leading to further complications. A much more numerically sound approach is to use *divided differences*.

Definition 1.6. Let $f[x_i] = f(x_i)$. Then we define divided differences recursively via

$$f[x_i, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}.$$

Proposition 1.7. *The divided differences are exactly the coefficients of the Newton interpolation:*

$$P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + \cdots + f[x_0, \ldots, x_n](x - x_0) \ldots (x - x_{n-1}).$$

Proving this is an exercise in algebra. Although this formulations looks very different from the Lagrange interpolation polynomial we found above, we know from linear algebra that they must be equivalent, and we are free to switch back and forth freely to use whichever format is convenient.

2 Error of interpolation

As always, these methods are meaningless unless we understand the error.

Theorem 2.1. *Let $x_0, \ldots, x_n \in [a, b]$ and f be $n + 1$ times differentiable on the interval $[a, b]$. Then for any $x \in [a, b]$, there is $\xi(x) \in [a, b]$ such that*

$$f(x) = P(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0) \ldots (x - x_n)$$

where P is the interpolating polynomial to f at $x_0, \ldots, x_n$.

The rough idea is to apply the MVT over and over again to $f - P$ to get an error term involving the $(n+1)^{st}$ derivative. First, contrast this with the error term for degree n Taylor approximation:

$$\frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)^{n+1}.$$

You should see that there are a lot of similarities. Since Taylor approximation uses n^{th} order information at x_0, we get $(x - x_0)^{n+1}$ in the error term for Taylor approximation. Since interpolation uses zeroth order information at each x_i, we get $(x - x_i)$ in the error term for polynomial interpolation.

To analyze the error further, we'll split it up into three pieces:

$$\frac{1}{(n+1)!} \quad f^{n+1}(\xi(x)) \quad (x - x_0) \ldots (x - x_n).$$

Question 2.2. Do these terms make sense? Can you come up with some justifications for the terms in the error?

1. The first term is a normalization term (same as Taylor series) and appears when you take derivatives of polynomials.
2. The second term says that if f is a polynomial of degree at most n, then we will approximate it exactly with no error. This is the error that comes from trying to approximate f using a polynomial. If f has a large $(n+1)^{st}$ derivative, then it may not be a good idea to use only a degree n polynomial approximation. (This is one situation where the Newton formulation is better than the Lagrange formulation – we can reuse most of the coefficients and only need to compute one more. However, the we need to recompute everything for the Lagrange interpolation.)
3. The third term says that if x is one of the interpolation points, then we will approximate it exactly with no error. This is the error that comes from the choice of interpolation points – if it is large at a specific point, then we get a poor approximation there. (More on this later.)

Question 2.3. Specifically in regards to the second point, is it better to use the Newton or Lagrange formulation?

Because we want to use polynomials, there's nothing we can do about the first two parts of the error. However, we have some control over the last term in the form of choosing our interpolation points. (For example, maybe functions evaluations are very expensive, so we want to design a sampling procedure to produce the best model.)

Remark 2.4. Sometimes, we won't even be able to do this. For example, stock market data comes with one data point per day, and there is nothing you can do to generate more data.

Before we get into a good way to choose the interpolation points, let's first investigate the impact that the interpolation points can have. Equally spaced points seem very natural to use. So let's take $x_i = i - 1$ for $1 \leq i \leq n$. If $x \approx n$, then

$$|(x-0)(x-1)\dots(x-n+1)| \approx n!$$

On the other hand, if $x \approx n/2$, then

$$|(x-0)(x-1)\dots(x-n+1)| \approx (n/2)!^2.$$

To compare these two, recall Stirling's approximation:

$$n! \approx \sqrt{2\pi n}(n/e)^n.$$

Applying this above,

$$\frac{n!}{(n/2)!^2} \approx \frac{\sqrt{2\pi n}(n/e)^n}{2\pi(n/2)(n/2e)^n} = \frac{2^{n+1}}{\sqrt{2\pi n}}.$$

When $n = 20$, this is about 187000, meaning that the approximation at the endpoints of the interval is about 187000 times worse than in the middle of the interval. This issue is known as *Runge's phenomenon*. Even for moderately sized intervals, this gets bad quickly, so we need to look for alternatives. Furthermore, it might seem like a good idea to use higher and higher degrees, but we can see from the above that this will increase the error of the interpolation points, which grows exponentially in n.

2.1 Chebyshev points

In order to massage the third term of the error

$$(x - x_0) \dots (x - x_n)$$

to obtain better performance, let's set a goal of looking for $x_0, \dots, x_n$ such that

$$|(x - x_0) \dots (x - x_n)| \leq 1$$

In other words, we want to find a polynomial with $n+1$ zeroes on $[a, b]$ (we'll use $[-1, 1]$ for convenience) that is bounded in absolute value by 1. It's not too difficult to do this for low degree polynomials. For example, when $n = 0$, we can pick $T_0(x) = 1$. When $n = 1$, we can pick $T_1(x) = x$. Already for $n = 2$ things get a bit trickier.

Question 2.5. Can you find a polynomial $T_2(x)$ that has 2 zeroes and is bounded in absolute value by 1 on $[-1, 1?$

There are quite a few options, but it turns out the one we want is $T_2(x) = 2x^2 - 1$. We can try to keep going like this, but it's going to get increasingly difficult unless we come up with another approach.

Chebyshev's clever idea was to investigate certain functions defined in terms of certain trigonometric functions. These are interesting for many reasons that are beyond this course, and are called *Chebyshev polynomials*. Let's try the substitution $x = \cos(t)$. The above polynomial turns into

$$T_2(\cos(t)) = 2\cos(t)^2 - 1 = \cos(2t).$$

This also works for T_0 and T_1 that we found above, suggesting the general formula

$$T_k(\cos(t)) = \cos(kt).$$

However, we run into an immediate issue: does this even define a polynomial?

Question 2.6. How can we recognize the T_k defined above as a polynomial?

Recall the formulas for the sum and difference of angles:

$$\begin{aligned} \cos(a + b) &= \cos(a)\cos(b) - \sin(a)\sin(b) \\ \cos(a - b) &= \cos(a)\cos(b) + \sin(a)\sin(b) \\ \cos(a + b) + \cos(a - b) &= 2\cos(a)\cos(b). \end{aligned}$$

Setting $a = kt$ and $b = t$, we get

$$\begin{aligned} \cos((k+1)t) + \cos((k-1)t) &= 2\cos(kt)\cos(t) \\ T_{k+1}(\cos(t)) + T_{k-1}(\cos(t)) &= 2T_k(\cos(t))T_1(\cos(t)) \end{aligned}$$

returning to the variable x

$$T_{k+1}(x) + T_{k-1}(x) = 2xT_k(x).$$

Rearranging, we can define the Chebyshev polynomials inductively by

$$T_{k+1} = 2xT_k - T_{k-1}.$$

Question 2.7. Compute and graph the first few Chebyshev polynomials.

We verified manually that the first few Chebyshev polynomials have the desired properties, but what about the rest? Let's set $T_k = 0$ and see what we get.

$$\begin{aligned} T_k(x) &= 0 \\ T_k(\cos(t)) &= 0 \\ \cos(kt) &= 0 \\ kt &= \frac{\pi}{2} + i\pi \qquad i = 0, 1, \ldots, (k-1) \\ t &= \frac{\pi}{2k} + \frac{i\pi}{k} \qquad i = 0, 1, \ldots, (k-1) \end{aligned}$$

Recall $\cos(t) = x$, so we get

$$x = \cos\left(\frac{\pi}{2k} + \frac{i\pi}{k}\right) \qquad i = 0, 1, \ldots, (k-1).$$

These are the interpolation points we want to use. In order to get these points for an arbitrary interval $[a, b]$, we can transform $[-1, 1]$ to $[a, b]$ by scaling and then shifting. (One way to think about this is that we took evenly spaced points on the unit circle from 0 to π and projected them down to the x-axis. This concentrates interpolation points near the endpoints. This gives a huge improvement on the error, especially at the endpoints, coming from the choice of interpolation points.

(Another alternative is to solve the recurrence relation explicitly to find $T_k(x) = \cos(kx)$.)

3 Other methods

The most obvious drawback of Chebyshev interpolation is that it's hard to apply: in most numerical situations, we will only have certain data available and we cannot freely choose the interpolation points.

Question 3.1. How can generalize the interpolation problem?

There are two other common approaches. One is to incorporate more information about the function if it is available. In particular, we want to use any available information about the derivatives of $f(x)$ in constructing an interpolating polynomial. The other is to break up the problem into many smaller intervals.

1. If we have additional information about derivatives of $f(x)$, then we can try to match not only the given points but also the derivatives at those points.

2. We've seen that higher degrees don't necessarily fix anything. For example, if we naively use the higher and higher degree Lagrange interpolations, we will get worse and worse performance near the endpoints.

3.1 Incorporating derivatives

In the interpolation problem, we only ask that $P(x_i) = f(x_i)$ for all the interpolation points. If we additionally have derivative information, then we can also ask that $P^{(k)}(x_i) = f^{(k)}(x_i)$ for $1 \leq k \leq m$. Since the derivative of a polynomial is another polynomial, each additional derivative adds a linear constraint on the coefficients (and also increases the degree by 1). The linear algebra is doable, but a lot trickier here than it was before without derivatives. Either way, we again expect that the interpolating polynomial will be unique if it exists.

Remark 3.2. While it's possible to work in general with mixed derivative information (e.g. m_i derivatives at x_i), we'll work in the specific case of *Hermite interpolation* where we want to match up to the m^{th} derivative at each interpolation point.

Question 3.3. Which formulation seems better for incorporating derivative information?

For the Lagrange perspective, like we did without derivatives, we'll want to pick a clever choice of functions so we can directly plug in the relevant coefficients. The first step here is the following criterion.

Question 3.4. Find a criterion for both a polynomial and its derivative to be zero at a point.

Using this criterion, we build polynomials of degree $(m+1)(n+1)-1$ which have the property that exactly one of the $P^{(k)}(x_i)$ is nonzero, while the rest are zero. Then we can do the same thing we did before and just add all these up with coefficients. We'll do this explicitly in the case of $m = 1$. In this case, we are looking for polynomials $A_{n,k}(x)$ and $B_{n,k}(x)$ of degree $2n+1$ such that

$$A_{n,k}(x_j) = \delta_{jk} \quad A'_{n,k}(x_j) = 0 \quad B_{n,k}(x_j) = 0 \quad B'_{n,k}(x_j) = \delta_{jk}.$$

The $B_{n,k}(x)$ functions are slightly easier to describe.

Question 3.5. How should we choose the $B_{n,k}(x)$?

The right choice here is

$$B_{n,k}(x) = (x - x_k)L_{n,k}(x)^2.$$

By definition of $L_{n,k}(x)$, we have $B_{n,k}(x_j) = 0$ for all j. Furthermore,

$$B'_{n,k}(x) = L_{n,k}(x)^2 + (x - x_k)2L_{n,k}(x)L'_{n,k}(x)$$

so this function also has the correct derivative behavior. The $A_{n,k}(x)$ are a bit trickier.

Question 3.6. How should we choose the $A_{n,k}(x)$?

We'll definitely want $L_{n,k}(x)^2$ to appear since $A_{n,k}(x)$ needs to be zero at all the $x_{j\neq k}$ and have derivative zero at all the x_j. Since we're looking for a polynomial of degree $2n+1$, we know that we just need to multiply by another linear factor:

$$(a + bx)L_{n,k}(x)^2.$$

We have two coefficients to identify, and two constraints to identify them:

$$A_{n,k}(x_k) = (a + bx_k)L_{n,k}(x_k)^2 = (a + bx_k) = 1$$

$$A'_{n,k}(x_k) = bL_{n,k}(x_k)^2 + 2(a + bx_k)L_{n,k}(x_k)L'_{n,k}(x_k) = b + 2(a + bx_k)L'_{n,k}(x_k) = 0$$

The first equation says $a + bx_k = 1$, so plugging that into the second yields

$$b + 2L'_{n,k}(x_k) = 0 \implies b = -2L'_{n,k}(x_k).$$

Plugging this back into the first equation gives

$$a = 1 + 2L'_{n,k}(x_k)x_k.$$

Overall, we obtain

$$A_{n,k}(x) = (1 + 2L'_{n,k}(x_k)x_k - 2L'_{n,k}(x_k)x)L_{n,k}(x)^2$$

which we may simplify slightly as

$$A_{n,k}(x) = (1 - 2L'_{n,k}(x_k)(x - x_k))L_{n,k}(x)^2.$$

The general procedure is similar, but gets quite messy as m increases.

It turns out the Newton formulation with divided differences is slightly more amenable to introducing derivatives (though both methods are viable), provided we carefully define the divided differences. Consider $f[x_0, x_1] = \frac{f(x_1)-f(x_0)}{x_1-x_0}$. If we let $x_1 \to x_0$, we get

$$\lim_{x_1 \to x_0} \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f'(x_0)$$

and we can use this to define $f[x, x]$. Similarly, although the technical details are a bit subtle, we can think of the divided difference $f[x, \ldots, x]$ (repeated k times) as an analogue of the $(k-1)^{st}$ derivative $f^{(k-1)}(x)$. Using this, we can think of incorporating derivative information by simply repeating points in the previous Newton formula.

As you might expect from the generalizations above (especially the Newton perspective of letting points "collide" or "flow together"), (and Taylor's error formula), the error for Hermite interpolation is given by

$$\frac{f^{(m+1)(n+1)}(\xi(x))}{((m+1)(n+1))!}(x - x_0)^m \ldots (x - x_n)^m.$$

and it splits up in the same way as before. Thus, it also makes sense to use Chebyshev points here as well.

3.2 Splines

Alternatively, no one is forcing us to use one polynomial for the entire interval we want to approximate. You have probably encountered the simplest kind of splines already: step functions and piecewise linear approximation. Given points $(x_i, f(x_i))$, we'll approximate f using the line segments between each $(x_i, f(x_i))$ and $(x_{i+1}, f(x_{i+1}))$.

Question 3.7. Is there any freedom in choosing the coefficients of the line segments?

If we have a lot of interpolation points, piecewise linear approximation can come very close to the actual function. However, we will lose any kind of smoothness that the function has. This is okay for integration (trapezoid rule!), but terrible for differentiation, because our function will not be differentiable at the interpolation points, which might be where we care most about accuracy. What happens when we increase the degree?

Question 3.8. Is there any freedom in choosing the coefficients if we instead use quadratic polynomials? More generally, how many degrees of freedom are there if we use a degree n polynomial?

On each interval, we have two direct constraints: we want our interpolating polynomial to pass through the two given endpoints. Other than that, there are no immediate constraints on the polynomials we use. Since a degree n polynomial has $n+1$ coefficients, this means that there are $(n-1)n$ degrees of freedom to work tih since there are n intervals. We can use this freedom to try to make our approximation smoother by matching derivatives at the interior points. One of the most common choices is the *cubic spline*, where we satisfy the following constraints:

1. On each interval $[x_i, x_{i+1}]$, we use a degree 3 polynomial q_i
2. $q_i(x_i) = f(x_i)$ and $q_i(x_{i+1}) = f(x_{i+1})$.
3. $q_i'(x_{i+1}) = q_{i+1}'(x_{i+1})$
4. $q_i'(x_{i+1}) = q_{i+1}'(x_{i+1})$

Note that the last two constraints have nothing to do with f' and f''. We are only asking for the function to be differentiable at the x_j, not to match the particular values of the function's derivative or second derivative, etc.

Question 3.9. How many constraints do the above conditions put on the coefficients?

There are two "leftover" degrees of freedom for q_0 and q_n. They are often specified as *boundary conditions* by one of the following

1. $q_0''(x_0) = q_n''(x_n) = 0$
2. $q_0'(x_0) = f'(x_0)$ and $q_n'(x_n) = f'(x_n)$.

It takes a bit of linear algebra, but writing each q_i as

$$q_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

and expanding all the constraints above gives a linear system of equations which one can show has a unique solution. Finally, the error bound for cubic spline approximations is

$$\frac{5}{384} \max_x |f^{(4)}(x)| \max_j (x_{j+1} - x_j)^4.$$

Essentially, we are just doing "standard" interpolation on each interval, so we should look for the interval which gives the largest error.