EECS 16B Designing Information Devices and Systems II
Fall 2019 Note: Polynomial Interpolation and Global Approximations

Overview

In your calculus courses, you have seen how we use Taylor expansions in the local neighborhood of a point to approximate a function locally using a polynomial. When you studied Taylor series, you saw that by adding higher and higher order terms to the series, you make the approximation better and better. We have seen in our discussions of linearization how we can often do quite well locally by just keeping to first or second degree approximations.

Sometimes however, we are not conceptually content with a local approximation. We want an approximation that is valid more globally — on a larger region of interest, potentially even "everywhere." Sometimes, a purely local perspective like Taylor expansion ends up defining something that is reasonable everywhere. (This is what, for example, let you define the exponential function and thus indirectly, sines and cosines.) While philosophically intriguing, from a practical engineering perspective, it is extraordinarily rare for a truly local view to extrapolate to everywhere — this essentially never happens when we try to learn from real data or samples of functions of interest.

One approach to global approximation is implicit — we don't actually actively approximate everywhere all at once, instead we build local approximations in the neighborhood of everywhere that we end up needing to approximate at all. This is often what is done when one is using local approximations in the service of iterative optimization — we use local approximations to take a step and then re-build our approximations where we end up.

But there is another approach to global approximation — parametric. Here, we have a function that is defined by parameters. We use data to learn the parameters, and then we can evaluate this function wherever we want to get a global approximation. The function together with the set parameter values defines the approximation. We will generally use linearly-parameterized functions in the same style that we have done elsewhere and was done in 16A.

A particularly interesting type of approximation is interpolating approximations. Here, we will be given a finite set of sample points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ that may be spaced broadly apart. Our task will be to produce a function $f(x)$ that passes through our sample points, but that we can query anywhere. Interpolation focuses our attention on certain core issues that always exist in all learning problems without having to worry about the quality of the fit to the training data — in interpolation, the fit is always perfect.

In this note, our motivation will be data collected in a time segment — for example, voltage data collected from a neural probe that we want to use to figure out what the person wants their cybernetic arm to do. We collect samples of this data over an interval of interest. If we can globally approximate this well with the need for fewer samples, it simplifies the system design outside of the computer while allowing any computation that would want more samples to generate those extra samples synthetically within the computer[1] by using

[1] The most viscerally understandable need for interpolation/approximation is provided by GPS, which you saw in 16A. GPS uses the distance to satellites to triangulate positions on Earth. Actual GPS systems would like to get position accuracies within

the approximation in place of additional data.

1 Polynomial Fitting

Let's say that we have m distinct points

$$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m).$$

Since we are assuming that these points are the evaluation of some unknown function $y = f(x)$, it is clear that all the x_i must be distinct.

What parametric function family should we use to approximate? We are familiar with polynomials from our study of Taylor series, etc., so why not use them?

We are searching for a polynomial that passes through all m of our points. Let this polynomial be

$$g(x) = a_0 + a_1 x + a_2 x^2 + \cdots,$$

where the a_i are constant coefficients to be determined. These a_i are the parameters that define the function. These are what we need to learn from the data.

The first question to ask is: what should the degree of our polynomial be? Intuitively, since we are trying to pass through m points, our polynomial should have at least m degrees of freedom, and so must have at least m coefficients: $a_0, \ldots, a_{m-1}$. Thus, it should be of degree at least $m-1$ to generically be able to interpolate.

As a reality check, we can consider the special case of $m = 2$. In that case, we know that a polynomial of degree $m - 1 = 1$ (i.e. a line) can pass through any two points with distinct x-coordinates, but that a polynomial of degree 0 (i.e. a constant function) cannot. So our claim sounds correct.

Note, however, that it still has not yet been rigorously justified — it is simply a guess, that we can work with and justify later. Assuming that such a polynomial exists, we can write what we know:

$$\begin{aligned}
y_1 &= g(x_1) = a_0 + a_1 x_1 + a_2 x_1^2 + \ldots + a_{m-1} x_1^{m-1} \\
y_2 &= g(x_2) = a_0 + a_1 x_2 + a_2 x_2^2 + \ldots + a_{m-1} x_2^{m-1} \\
&\vdots \\
y_m &= g(x_m) = a_0 + a_1 x_m + a_2 x_m^2 + \ldots + a_{m-1} x_m^{m-1}.
\end{aligned}$$

Stacking the above equations and rearranging into standard 16-style matrix-vector form, we obtain

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{m-1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_m & x_m^2 & \cdots & x_m^{m-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}.$$

Recall that the x_i and y_i are *known* quantities, and the a_i are unknowns. So the above equation is really just

meters. However, as you saw in 16A, GPS uses maximum correlation to figure out the delay (and hence distance) to the satellites at known positions. The speed of light is 3×10^8 meters per second; and this means that the naive approach would require taking data every 10ns to get 3m accuracy for each distance measurement. Real-world GPS systems effectively take at least 25 times fewer samples than this and use approximation-style ideas to get a very fine handle on the delay of maximum correlation without having to take as many samples. This is but one example.

a linear system of equations expressed in matrix form. Giving names to things, we can define

$$D = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{m-1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_m & x_m^2 & \cdots & x_m^{m-1} \end{bmatrix}$$
$$\vec{a} = \begin{bmatrix} a_0 & a_1 & \cdots & a_{m-1} \end{bmatrix}^T$$
$$\vec{y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix}^T,$$

with our task being to solve the linear system

$$D\vec{a} = \vec{y}$$

for $\vec{a}$.

2 Showing existence and uniqueness

Ideally, we'd able to show (1) that a solution for $\vec{a}$ always exists, and that (2) such a solution is unique. Then, we'd have shown that polynomials of degree at most $m-1$ are able to fit all sets of m points with distinct x-coordinates, and that we cannot do so using only polynomials of degree at most $m-2$. From 16A, we know that we can show both of these conditions if and only if the square matrix D is of full rank — i.e. invertible.

Can we do this? Looking at its entries, we see that D has a special structure. Each of its columns contain the evaluation of a monomial at a set of points - specifically, the ith column contains the evaluation of the monomial x^{i-1} at $x_1, x_2, \ldots, x_m$. Because of this structure, D is known as a *Vandermonde matrix* in the literature.

How can we prove that this matrix is of full rank? One way is to compute its determinant and show that it is nonzero. This is a valid approach, and indeed can be made to work, with some effort[2]. Alternatively, if we can show that its m columns form a basis for an m-dimensional vector space, then they must all be linearly independent. For instance, if we could produce a matrix B such that

$$DB = I,$$

then it would be immediately clear that D is invertible. This is the approach we will take.

What would such a B look like? Let's write it down, as

$$B = \begin{bmatrix} B[1][1] & B[1][2] & \cdots & B[1][m] \\ B[2][1] & B[2][2] & \cdots & B[2][m] \\ \vdots & \vdots & \ddots & \vdots \\ B[m][1] & B[m][2] & \cdots & B[m][m] \end{bmatrix}.$$

[2] Indeed, it is the most common approach used in other courses and at other schools — however, we strongly prefer changing coordinates to blind computation and so we will proceed differently.

Then we see that the ith column of DB becomes

$$(DB)_i = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{m-1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_m & x_m^2 & \cdots & x_m^{m-1} \end{bmatrix} \begin{bmatrix} B[1][i] \\ B[2][i] \\ \vdots \\ B[m][i] \end{bmatrix} = \begin{bmatrix} B[1][i] + B[2][i]x_1 + B[3][i]x_1^2 + \ldots + B[m][i]x_1^{m-1} \\ B[1][i] + B[2][i]x_2 + B[3][i]x_2^2 + \ldots + B[m][i]x_2^{m-1} \\ \vdots \\ B[1][i] + B[2][i]x_m + B[3][i]x_m^2 + \ldots + B[m][i]x_m^{m-1} \end{bmatrix}.$$

Defining the polynomial

$$L_i(x) = B[1][i] + B[2][i]x + B[3][i]x^2 + \ldots + B[m][i]x^{m-1},$$

we can rewrite

$$(DB)_i = \begin{bmatrix} L_i(x_1) \\ L_i(x_2) \\ \vdots \\ L_i(x_m) \end{bmatrix}.$$

Since $DB = I$, its ith column must equal the ith identity basis vector, where all the entries except the ith entry equal zero, and the ith entry equals 1. Thus, we have that $L_i(x_i) = 1$ and that, for all $j \neq i$, $L_i(x_j) = 0$.

Do such polynomials exist? Let's first try to enforce the first condition - that for all $i \neq j$, $L_i(x_j) = 0$. Thus, $(x - x_j)$ must be a factor of $L_i(x)$. Multiplying these factors together for all $j \neq i$, we obtain

$$a_i(x) = (x - x_1)(x - x_2)\cdots(x - x_{i-1})(x - x_{i+1})\cdots(x - x_m).$$

Is $a_i(x)$ the polynomial we want? Not quite. When we set $x = x_i$, we get

$$a_i(x_i) = (x_i - x_1)(x_i - x_2)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_m),$$

which may not equal to 1, which is what we want it to equal. However, what is crucial to notice is that $a_i(x_i) \neq 0$, since it is the product of $m-1$ *nonzero* terms. Thus, we can simply divide $a_i(x)$ by $a_i(x_i)$, to obtain

$$L_i(x) = \frac{a_i(x)}{a_i(x_i)} = \frac{(x - x_1)(x - x_2)\cdots(x - x_{i-1})(x - x_{i+1})\cdots(x - x_m)}{(x_i - x_1)(x_i - x_2)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_m)}.$$

→ This is a technique called Lagrange Interpolation in CS70.

We can instantly see that, for $i \neq j$,

$$L_i(x_j) = \frac{a_i(0)}{a_i(x_i)} = \frac{0}{a_i(x_i)} = 0,$$

as desired, and that

$$L_i(x_i) = \frac{a_i(x_i)}{a_i(x_i)} = 1,$$

which is also as desired. So this $L_i(x)$ has all the properties that we want!

So, from $L_i(x)$, can we obtain the factors $B[\cdot][i]$? Recall that we further require $L_i(x)$ to be a polynomial of degree at most $m-1$, since it only has nonzero coefficients for the x^0 through x^{m-1} terms. But looking at our constructed $L_i(x)$, we see that it is indeed of degree exactly $m-1$ since it is the product of $(m-1)$ terms $(x - x_j)$, so it's what we're looking for!

Thus, expanding out $L_i(x)$, we know we can write it as

$$L_i(x) = B[1][i] + B[2][i]x + B[3][i]x^2 + \ldots + B[m][i]x^{m-1}$$

and thereby, simply expanding out the polynomial lets us determine the values of the $B[\cdot][i]$. Similarly, by computing the various other $L_j(x)$, we can determine all the entries of the matrix B. And then, since

$$DB = \begin{bmatrix} L_1(x_1) & L_2(x_1) & \cdots & L_m(x_1) \\ L_1(x_2) & L_2(x_2) & \cdots & L_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ L_1(x_m) & L_2(x_m) & \cdots & L_m(x_m) \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} = I,$$

we have shown that D is full rank! Therefore, we know that our equation $D\vec{\alpha} = \vec{y}$ will always have a unique solution, so our polynomial interpolation technique will always work.

3 Least Squares

We now know how to construct a degree-$(m-1)$ polynomial that passes through any given m points with distinct x-coordinates. What if we want to fit a function using a polynomial of lower degree - say, degree at most k? Clearly, such a polynomial may not be able to pass through all of the points exactly. But if we're OK with that, how can we find a polynomial that passes "near" our points with a minimum of error?

Essentially, this is a generalization of the technique we saw in 16A where we used least squares to fit a line through a set of points. In both cases, given points $(x_1, y_1), \ldots, (x_m, y_m)$, we are interested in finding an approximating function $g(x)$ that minimizes the error

$$e = \sum_{i=1}^{n} (y_i - g(x_i))^2.$$

We are interested in finding a degree-k polynomial $g(x)$ that minimizes this error. Note that $k \leq m-1$, since we can fit all of our points exactly with polynomials of degree $m-1$, so there is never conventionally[3] a need for a polynomial of a higher degree.

Letting

$$g(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \ldots + \alpha_k x^k,$$

we see that

$$\begin{bmatrix} g(x_1) \\ g(x_2) \\ \vdots \\ g(x_m) \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^k \\ 1 & x_2 & x_2^2 & \cdots & x_2^k \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_m & x_m^2 & \cdots & x_m^k \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix},$$

[3] The next note on the DFT will help you understand this better.

using a similar rearrangement to what we did previously. Now, for convenience, let

$$D_k = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^k \\ 1 & x_2 & x_2^2 & \cdots & x_2^k \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_m & x_m^2 & \cdots & x_m^k \end{bmatrix}$$
$$\vec{\alpha} = \begin{bmatrix} a_0 & a_1 & \cdots & a_k \end{bmatrix}^T$$
$$\vec{y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix}^T,$$

again in analogy to before.

So the error term that we are trying to minimize can be written as

$$e = \|\vec{y} - D_k\vec{\alpha}\|^2.$$

We know that least squares lets us find a unique solution for $\vec{\alpha}$ that minimizes that term - specifically,

$$\vec{\alpha} = (D_k^T D_k)^{-1} D_k^T \vec{y}.$$

However, such a unique solution only exists if the columns of D_k are linearly independent. Can we show that this is always the case?

It turns out that we have already done the hard work for this. Recall that we have already shown above that the columns of

$$D = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{m-1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_m & x_m^2 & \cdots & x_m^{m-1} \end{bmatrix}$$

are linearly independent, by proving that the square matrix D is of full rank. Now, notice that the columns of D_k are simply the first k columns of D, which are linearly independent (as they are a subset of the m columns of D, which are all linearly independent). Thus, D_k has k linearly independent columns, so least squares will give us a unique $\vec{\alpha}$ that minimizes the error of our degree-k polynomial approximation.

Ultimately, we have shown how to construct a polynomial interpolation through a set of points, whether a degree $m-1$ interpolation that passes through all m of our points exactly, or a degree-k approximation that minimizes the error between our interpolation and the data points.

Once we have the coefficients α that represent such a polynomial, we can evaluate it anywhere we want at our leisure.

Contributors:

- Rahul Arya.
- Anant Sahai.

