1
Function Approximation

1.1 Introduction

In this chapter we discuss approximating functional forms. Both in econometric and in numerical problems, the need for an approximating function often arises. One possibility is that one has a finite set of data points and wants to determine the underlying functional form. For example, suppose one knows that next period's value of the interest rate, r_{t+1}, is some function of the current interest rate, but one doesn't know what the functional form is. That is, one has the empirical relationship

$$r_{t+1} = f(r_t) + \epsilon_{t+1} \tag{1.1}$$

and the question is then whether with enough data one can discover the functional form of f.

The need for an approximating functional form also arises if one could in principle the function value for any given set of arguments but that it is very expensive to do so. For example, the function value may be the outcome of many complex calculations and it may take a lot of computing time to calculate one function value. With an approximating functional form one could obtain (approximate) function values much quicker. Again the goal would be to come up with an approximating functional form using a finite set of data points. There is a big difference with the problem that the econometrician faces, however, because the econometrician cannot choose his data points. We will see in this section that the freedom to choose the

location of the arguments makes it much easier to come up with accurate approximations.

Finally, the theory on function approximation is very useful if one is trying to solve for a function that is (implicitly) defined by a system of functional equations.

1.2 Polynomial approximations

Most of this chapter will be devoted to polynomial approximations, i.e.,

$$y_t = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n, \tag{1.2}$$

where x is a scalar. Later in this chapter will we discuss functions with multiple arguments. We will see that there are actual many different types of polynomials by choosing different *basis functions*, $T_j(x)$. That is, a more general way to write polynomials is

$$y_t = a_0 + a_1 T_1(x) + a_2 T_2(x) + \cdots + a_n T_n(x). \tag{1.3}$$

For example, $T_j(x)$ could be $(\ln(x))^j$. Moreover, one can take transformations. For example, if one knows that the function value is always positive, one could use this information by letting

$$y_t = \exp(a_0 + a_1 T_1(x) + a_2 T_2(x) + \cdots + a_n T_n(x)). \tag{1.4}$$

1.2.1 How good are polynomial approximations

Weierstrass theorem tells us that a continuous real-valued function defined on a bounded interval on the real line can be approximated arbitrarily well using the sup norm if the order of the polynomial goes to infinity. Even functions that have a discontinuity can be approximated arbitrarily well if one uses another norm then the sup norm.

1.2.2 How to find polynomial approximations

In this subsection, we discuss four different ways to come up with a polynomial approximation. The procedures differ in whether they use only local information (Taylor expansion) and whether they use information about derivatives or not.

Taylor expansion

If one has the function value and n derivatives at one point, x_0, then one can calculate a polynomial approximation using the Taylor expansion.

$$f(x) \approx f(x_0) + (x - x_0) \left.\frac{\partial f(x)}{\partial x}\right|_{x=x_0} + \cdots + \frac{(x - x_0)^n}{n!} \left.\frac{\partial^n f(x)}{\partial x^n}\right|_{x=x_0} \tag{1.5}$$

Note that the right-hand side is indeed a polynomial.

Projection

More common though is that one has $n+1$ function values, f_0, f_1, $\cdots$, f_n, at $n+1$ arguments, x_0, x_1, $\cdots$, x_n. There are two procedures to find the coefficients of the polynomial. The first is to run a regression. If you do it right you get an R^2 equal to 1.

Lagrange Interpolation

The approximating polynomial is also given by

$$f(x) \approx f_0 L_0(x) + \cdots + f_n L_n(x), \tag{1.6}$$

where

$$L_i(x) = \frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}. \tag{1.7}$$

$L_i(x)$ is a polynomial so the right-hand side of 1.6 is a polynomial as well. So we only have to show that this polynomial gives an exact fit at the $n+1$ points. This is easy if one realizes that

$$L_i(x) = \begin{cases} 1 \text{ if } x = x_i \\ 0 \text{ if } x \in \{x_0, \cdots x_n\} \setminus \{x_i\} \end{cases} . \tag{1.8}$$

Consequently, one gets

$$f(x_i) = f_i \tag{1.9}$$

It is unlikely that you will actually find it useful to write the polynomial like this, but we will see that this way of writing a polynomial is useful for numerical integration.

Hermite Interpolation

The last two procedures only used function values at a set of grid points. Now suppose that in addition to $n+1$ function values one also has numerical values for the derivatives at the nodes, i.e., f_0', f_1',$\cdots$, f_n'. With these $2(n+1)$ pieces of information one can calculate a $(2n+1)^{\text{th}}$ order polynomial.

$$f(x) \approx \sum_{i=0}^{n} f_i H_i(x) + \sum_{i=0}^{n} f_i' \widetilde{H}_i(x), \tag{1.10}$$

where

$$H_i = \left(1 - 2L_i'(x_i)(x-x_i)\right) L_i(x)^2$$
$$\widetilde{H}_i = (x-x_i) L_i(x)^2$$

Note that

$$H_i(x) = \widetilde{H}_i'(x) = \begin{cases} 1 \text{ if } x = x_i \\ 0 \text{ if } x \in \{x_0, \cdots x_n\} \setminus \{x_i\} \end{cases}$$

$$\widetilde{H}_i(x) = H_i'(x) = \begin{cases} 0 \text{ if } x = x_i \\ 0 \text{ if } x \in \{x_0, \cdots x_n\} \setminus \{x_i\} \end{cases}$$

The approximation gets the function values right at all nodes because the $\widetilde{H}_i(x_j)$ terms are all zero and the $H_i(x_j)$ terms are 1 at $x_j = x_i$ and zero otherwise. The approximation gets the derivatives right because the $H_i(x_j)$ are all zero and the $\widetilde{H}_i(x_j)$ select with its zero/one structure the appropriate derivative.

1.2.3 Orthogonal polynomials

From the discussion above, it became clear that finding the coefficients of the polynomial is like a projection on the space spanned by the basis function, i.e., like a regression. In any introductory econometrics course one learns about the problem of multicollinearity and the advantage of having uncorrelated explanatory variables. The same is true in function approximation. Moreover, in numerical problems it is important to have good initial conditions. The problem with the basis functions of regular polynomials (i.e., $1, x, x^2$, etc.) is that these terms are often highly correlated unless one has a lot of variation in the argument. Adding one additional polynomial term could thus very well mean that the coefficients of all polynomial terms change substantially even if the extra terms has little additional explanatory power.

Orthogonal polynomials are such that the basis functions are by construction orthogonal with respect to a certain measure. That is, the basis functions of all orthogonal polynomials satisfy

$$\int_a^b T_i(x)T_j(x)w(x)dx = 0 \quad \forall i, j \ \ni i \neq j \tag{1.11}$$

for some weighting function $w(x)$. Popular orthogonal polynomials are Chebyshev polynomials and the reason they are popular will become clear below. Chebyshev polynomials are defined on the interval $[-1, 1]$ and the weighting function is given by

$$w(x) = \frac{1}{(1-x^2)^{1/2}}. \tag{1.12}$$

The basis functions of the Chebyshev polynomials are given by

$$\begin{aligned} T_0^c(x) &= 1 \\ T_1^c(x) &= x \\ T_{i+1}^c(x) &= 2xT_i^c(x) - T_{i-1}^c(x) \ \ i > 1 \end{aligned}$$

Note that if one builds a polynomial with Chebyshev basis functions, i.e.,

$$f(x) \approx \sum_{j=0}^{n} a_0 T_j^c(x), \tag{1.13}$$

then one also has a standard polynomial, i.e., of the form

$$b_0 + b_1 x + b_2 x^2 + \cdots + b_n x^n,$$

where the b_js are functions of the a_js. The same is true for other orthogonal polynomials. So the reason we use orthogonal polynomials is not that we get a different type of product. The reason we use orthogonal polynomials is that it is easier to find the coefficients, for example, because if one add higher order terms good initial conditions are the coefficients one found with the lower-order approximation.

Chebyshev polynomials are defined on a particular interval but note that a continuous function on a compact interval can always be transformed so that it is defined in this range.

1.2.4 Chebyshev nodes

We now define a concept of which the importance will become clear in the remainder of this section. Chebyshev nodes are the x-values at which a basis function is equal to zero. For example,

$$T_2^c = 2x^2 - 1 \tag{1.14}$$

and the corresponding roots are equal to $-1/2$ and $+1/2$. Similarly,

$$T_3^c = 4x^3 - 3x \tag{1.15}$$

and the roots are are equal to $-\sqrt{3/4}$, 0, and $\sqrt{3/4}$. If one wants to construct n chebyshev nodes one thus takes the n^{th} Chebyshev basis function and finds the roots that set it equal to zero.

1.2.5 Uniform convergence

Weierstrass theorem implies that there are polynomial approximations that converge uniformly towards the true function, because convergence in the sup norm implies uniform convergence. To find this sequence, however, one must be smart in choosing the points that one uses in the approximation. If one uses observed data points one doesn't have this degree of freedom, but in many numerical problems one does. The flexibility to choose the approximation points will turn out to be a great benefit in many numerical problems.

It turns out that by fitting the polynomial at the Chebyshev nodes guarantees uniform convergence. A famous function to document how terrible not having uniform convergence can be is:

$$f(x) = \frac{1}{1 + x^2}$$

defined on $[-1, 1]$. As an exercise you should compare the following two strategies to find the coefficients of the approximating polynomial. The first strategy finds the coefficients of the approximating polynomial using $n + 1$ equidistant points and its function values. The second strategy uses Chebyshev nodes. The polynomials that one obtains with equidistant points only converge point wise and as n increases one sees bigger and bigger oscillations.

For a formal discussion one should read Judd (1998). But some intuition of why Chebyshev nodes are so powerful can be obtained by thinking of the formula for standard errors in a standard regression problem. If X is the matrix with all the observations of the explanatory variables and σ^2 the error variance, then the standard error is given by $\sigma^2(X'X)^{-1}$. That is, the further apart the x-values the smaller the standard error. Chebyshev nodes are more spread towards the boundaries then equidistant points and, thus, we obtain a more accurate approximation using polynomials fitted at Chebyshev nodes.

1.2.6 Other types of basis functions

Orthogonal polynomials can be written as ordinary polynomials. They differ from ordinary polynomials by having different basis functions, but an n^{th}-order Chebyshev polynomial can be written as an n^{th}-order regular polynomial. Nevertheless one has quite a bit of flexibility with polynomial approximations. For example, instead of approximating $f(x)$ one can approximate $f(\exp \widetilde{x}) = f(\exp(\ln x))$. Or if one knows that the function value is always positive one can approximate $\ln(f(x))$.

Of course, one also could consider alternatives to using polynomial basis functions. An alternative is to use neural nets. The idea behind neural nets is very similar to using polynomial approximations but they use different basis functions to build up the approximating function. In particular let $\boldsymbol{x}$ be a vector with function arguments and let $f : R^n \to R$. Then the neural net approximation is given by

$$f(\boldsymbol{x}) \approx \sum_{j=1}^{J} \gamma_j g(w_j' \boldsymbol{x} + b_j), \tag{1.16}$$

where $w^j \in R^n$, γ_j, $b_j \in R$, and $g : R \to R$ is a scalar squashing function, that is, a function with function values in the unit interval. Neural net approximations are not very popular in macroeconomics. The reason is that neural net approximations need quite a few parameters (layers) to approximate low-order polynomials and many series in economics are well approximated with polynomials. Neural nets have been more successful in explaining time series with more chaotic behavior.

1.3 Splines

The approximation procedures discussed above all had in common that for each possible argument at which one would like to evaluate the function, there is one identical polynomial. The idea about splines is to split up the domain into different regions and to use a different polynomial for each region. This would be a good strategy if, the function can only be approximated well with a polynomial of a very high order over the entire domain, but can be approximated well with a sequence of low-order polynomials for different parts of the domain. The inputs to construct a spline are again $n+1$ function values at $n+1$ nodes.

Splines can still be expressed as a linear combination of basis functions which is the same for each possible argument, but this is not a polynomial. The basis functions are zero over most of the domain. Thus splines take a much more local approach and a change in a function value far away from x_i is much less likely to affect the approximation for $f(x_i)$ when using splines then when using polynomial approximations.

Piece-wise linear

The easiest spline to consider is a piecewise linear interpolation. That is for $x \in [x_i, x_{i+1}]$

$$f(x) \approx \left(1 - \frac{x - x_i}{x_{i+1} - x_i}\right) f_i + \left(\frac{x - x_i}{x_{i+1} - x_i}\right) f_{i+1}.$$

n^{th}-order spline

Piece-wise linear splines are in general not differentiable at the nodes and this could be a disadvantage. But it is easy to deal with this by fitting a (low-order) polynomial on each segment and choose the coefficients such that it fits the function values at the nodes and the function is smooth at the nodes. Consider what needs to be done to implement a cubic spline. A cubic spline uses

$$f(x) \approx a_i + b_i x + c_i x^2 + d_i x^3 \text{ for } x \in [x_{i-1}, x_i].$$

Since we have n segments we have n separate cubic approximations and thus $4n$ unknown coefficients. What are the conditions that we have to pin down these coefficients?

- We have $2 + 2(n-1)$ conditions to ensure that the function values correspond to the given function values at the nodes. For the two endpoints, x_0 and x_{n+1}, we only have one cubic that has to fit it correctly. But for the intermediate nodes we need that the cubic approximations of both adjacent segments give the correct answer. For

example, we need that

$$\begin{aligned} f_1 &= a_1 + b_1 x_1 + c_1 x_1^2 + d_1 x_1^3 \text{ and} \\ f_1 &= a_2 + b_2 x_1 + c_2 x_1^2 + d_2 x_1^3 \end{aligned}$$

- To ensure differentiability at the intermediate nodes we need

  $b_i x_i + 2c_i x_i + 3d_i x_i^2 = b_{i+1} x_i + 2c_{i+1} x_i + 3d_{i+1} x_i^2$ for $x_i \in \{x_1, \cdots, x_{n-1}\}$,

  which gives us $n-1$ conditions.

With a cubic spline one can also ensure that second derivatives are equal. That is,

$$b_i + 2c_i + 6d_i x_i = b_{i+1} + 2c_{i+1} + 6d_{i+1} x_i \text{ for } x_i \in \{x_1, \cdots, x_{n-1}\}.$$

We now have $2 + 4(n-1) = 4n - 2$ conditions to find $4n$ unknowns. So we need two additional conditions. For example, one can set the derivatives at the end points equal to zero. There are several algorithms to find the coefficients of spline approximations. See for example the approximation tool kit of Miranda and Fackler or the routines of Chapter 6 of Ken Judd's website at http://bucky.stanford.edu/numericalmethods/PUBCODE/DEFAULT.HTM

Coefficients of the spline

What are the coefficients of the spline? There are two answers. The first answer is to say that the coefficients that determine the functional form are the $n+1$ combinations for x_i, f_i. The other answer is to say that they are all the coefficients of the separate polynomials. Both answers are, of course, correct. But for latter applications it is more convenient to think of the coefficients of the spline as the (x_i, f_i) pairs and when the nodes don't change, then the coefficients are just the $n+1$ function values, i.e., the f_is. Now note that these values may not very directly reveal the function value at an arbitrary x, but given that we do use a particular spline, these function values pin down the spline and thus the approximating function value at x. More importantly, if finding the approximating function is part of a bigger numerical project then the goal is to find the $n+1$ function values at the nodes. Those completely pin down the solution.

1.4 Shape-preserving approximations

Polynomial approximations oscillate around the true function value. Moreover, these oscillations could be such that the approximating function does not inherit important properties of the approximating function, such as monotonicity or concavity. Can you approximate a function and preserve

such properties? That is suppose that the $n+1$ function values satisfy properties such as being positive, monotonicity, and concavity. Can you come up with an approximation that also has these properties?

If one uses one polynomial for the complete domain then this is more likely to happen if one uses a low-order polynomial. But since the fit in terms of distance may be worse for the low-order polynomial one could face a trade-off between accuracy and desired shape.

Actually, there is one approximation we discussed that automatically preserves monotonicity and concavity and that is the piece-wise linear approximation. That is, if the function $f(x)$ is monotonic and concave, then the $n+1$ function values will of course inherit these properties and so will the interpolated values. Schumacher's algorithm find's second-order splines that preserve (if present) monotonicity and concavity/convexity.

1.5 Multi-variate polynomials

1.5.1 Polynomial approximations

Extending polynomial approximations to multi-variate problems is very easy. Just like one easily formulates multivariate polynomials using standard basis functions, one can also formulate multivariate polynomials using other basis functions. For example, consider a function that has x and y as arguments. Then the n^{th}-order *complete* polynomial is given by

$$\sum_{i+j\leq n} T_i(x)T_j(y).$$

The n^{th}-order *tensor-product* polynomial is given by

$$\sum_{i\leq n, j\leq n} T_i(x)T_j(y).$$

1.5.2 Splines

Linear interpolation is easy for multivariate problems. Here I give the formulas for the interpolation of a function that depends on x and y and one has the four function values, f_{xy}, for the (x, y) combinations equal to (a, c), (b, c), (b, d), and (c, d). In this rectangle the interpolated value is given by

$$\begin{aligned} f(x,y) \approx \quad & 0.25 * \left(2 - 2\tfrac{x-a}{b-a}\right)\left(2 - 2\tfrac{y-c}{d-c}\right) f_{ac} \\ & +0.25 * \left(2\tfrac{x-a}{b-a}\right)\left(2 - 2\tfrac{y-c}{d-c}\right) f_{bc} \\ & +0.25 * \left(2\tfrac{x-a}{b-a}\right)\left(2\tfrac{y-c}{d-c}\right) f_{bd} \\ & +0.25 * \left(2 - 2\tfrac{x-a}{b-a}\right)\left(2\tfrac{y-c}{d-c}\right) f_{ad} \end{aligned} \tag{1.17}$$

Since the right-hand side has the cross product of x and y this is a first-order tensor but not a first-order complete polynomial. At the boundaries, i.e., the walls of the box, the function is linear, which means that the function automatically coincides with the interpolated values of the box right next to it.

Extending splines to multivariate problems is tricky. To see why think about what one has to ensure to construct a two-dimensional spline. The equivalent of the segment for the univariate case is now a box with the floor representing the space of the two arguments. Now focus on the fitted values on one of the walls. These function values and the corresponding derivatives have to be equal to those on the wall of the box next to it. So instead of enuring equality at a finite set of nodes one now has to ensure equality at many points even in the two-dimensional case.

1.6 What type of approximation to use?

Before one uses any approximation method one should ask oneself what one knows about the function. For example, it is possible that there are special cases for which one knows the functional form. Also, sometimes one knows that the functional form is more likely to be simple in the logs than in the original arguments. This would be the case if one expects the elasticity of f with respect to x to be fairly constant (as opposed to $\partial f/\partial x$.

The big question one faces is whether one should use one (possibly high-order) polynomial for the entire domain or several (possibly lower-order) polynomials for separate segments. The latter doesn't mean one has to use splines with many segments. Using prior knowledge one can also split the domain in a small number of separate regions, for example, a region where a borrowing constraint is and one where it is not binding. For each of the regions one than fits a separate approximating function. Note that for the case of borrowing constraints one typically wouldn't want the function at the node that connects the two regions to be differentiable. So one really could simply fit two polynomials on the two regions.