ECON 512: Empirical Methods **Fall 2018/Spring 2019**

Lecture 3: Function Interpolation and Continuous State DP

Instructor: Paul Grieco *Date: Jan 28*

These lecture notes are for my personal use. If you are reading them, I decided to distribute them as an experiment. There are typos and probably outright errors, if you find them please accept my apologies and report them to me.

These notes cover the basics of function interpolation and dynamic programing with a continuous state variable. For clarity they focus on the one-dimensional case. Here are references which go deeper into the subject:

- Judd, Chapter 6
- Miranda and Fackler, Chapter 6

3.1 Function Interpolation

3.1.1 Basic Idea

Idea: A (continuous) function may be hard to calculate, so approximate it with on that is easy to calculate using data from a small number of points.

How: Represent unknown function as a linear combine nation of known, tractable *basis functions*, $\phi_1, \phi_2, \ldots$. E.g., we will approximate f with,

$$\hat{f}(x) = \sum_{j=1}^{N} c_j \phi_j(x)$$

Where $c_1, \ldots, c_N$ are the basis coefficients to be determined. They are what will make the approximation finite dimensional.

Of course, the whole point is we don't want to evaluate $f(\cdot)$ everywhere, so we will instead do so at a small set of basis nodes.

There are two common choices for basis functions:

1. Polynomials: People often use traditional polynomials as the basis $1, x, x^2, \ldots$. However these can lead to multi-colinearity, orthogonal polynomials may work better. We will discuss these below.
2. Splines: A spline function is a piecewise function where the pieces are polynomials. They may provide better local properties as they only use "nearby" data to interpolate. We won't discuss them here beyond saying the are an alternative way to form a basis.

How good is the approximation? Every continuous function in the space spanned by the basis can be represented exactly using interpolation on the basis. This space grows richer as we increase the number of basis functions.

3.1.2 Interpolation is Solving a Linear System

In the end, interpolation is a lot like least squares estimation, with two key differences:

1. We get to choose where to evaluate the function, rather than being given data.
2. We know the exact answer where we evaluate the function, so no error terms. We'll have exactly as many "observations" as coefficients.

Given the basis $\{\phi_1, \ldots, \phi_N\}$ evaluate f at N nodes $(x_1, \ldots x_N)$ to find $y_i = f(x_i)$, then solve:

$$\begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_N) \end{bmatrix} = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_N(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_N(x_2) \\ \vdots & & & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \cdots & \phi_N(x_N) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix}.$$

Or in matrix notation with $y_i = f(x_i)$,

$$y = \Phi c.$$

So we can solve for the coefficients,

$$c = \Phi^{-1} y.$$

Of course, we are *interpolating*, there is no guarentee this approximation is any good outside the nodes.

3.2 Orthogonal Polynomials

In principle any set of polynomials can form a basis, the most obvious is

$$\{1, x, x^2, x^3, \ldots\}$$

However, while Weirstrauss's approximation theorem says that a function can be approximated arbitrarily well by a polynomial, it doesn't say which set of polynomials will work well. In fact, some polynomial systems work less well as the degree of polynomial increases.

Let's look at a quick example: Runge's phenomenon. See `RungeEx.m` in the repository. The problem is that equally spaced nodes and the high degree of polynomials make it hard to match the tails of the function.

One way to deal with this is to use a set of functions and nodes designed to have "good" properties in practice. We'll use orthogonal (specifically Chebyshev) polynomials.

A family of polynomials $\{\phi_n(x)\}$ is mutually orthogonal with respect to a weighting funciton $w(\cdot)$ on the interval $[a, b]$ if $\forall i, j$:

$$\langle \phi_i, \phi_j \rangle = \int_a^b \phi_i(x)\phi_j(x)w(x)dx = 0$$

Clearly, this will help with some of the multicoliniarity which is at the heart of Runge's phenomenon. We will focus on a particular family of orthogonal polynomials, Chebyshev polynomials, which approximate function on the interval $[-1,1]$. They are orthogonal with weighting function $w(x)=(1-x^2)^{\frac{1}{2}}$,

$$\begin{aligned}T_0(x) &= 1\\ T_1(x) &= x\\ T_2(x) &= 2x^2-1\\ T_3(x) &= 4x^3-3x\\ T_j(x) &= 2xT_{j-1}(x)-T_{j-2}(x)\end{aligned}$$

Which can also be defined as:

$$T_j(x)=\cos(j\cos^{-1}(x))$$

The key features of Chebyshev polynomials is that they can give us a bound on approximation error which can be minimized via choosing approximation nodes. They can be visualized using the script `chebPic.m`

Theorem 3.1 *(Judd, p. 222) Suppose that a function $f:[-1,1]\to R$ is C^k for some $k\geq 1$, and let I_n be the degree n polynomial interpolation of f based at the zeros of $T_n(x)$. Then*

$$||f-I_n||_\infty \leq \left(\frac{2}{\pi}\log(n+1)+1\right)\frac{(n-k)!}{n!}\left(\frac{\pi}{2}\right)^k\left(\frac{b-a}{2}\right)^k||f^{(k)}||_\infty$$

We refer to the zeros of $T_n(x)$ (the highest order polynomial used in the basis) as the Chebyshev nodes. You can calculate these zeros yourself, but they are available in various books.

The upshot of this is:

- The choice of nodes matters, and there is guidance on how to choose the best nodes for function interpolation.
- Smoother functions will be easier to approximate than less smooth, since the bound is governed by the derivative of f.

The Chebyshev nodes for T_n on $[-1,1]$ can be calculated as:

$$z_k=-\cos\left(\frac{2k-1}{2n}\pi\right)$$

You'll frequently want to shift these to the interval $[a,b]$,

$$x_k=\frac{a+b}{2}+\frac{a-b}{2}\cos\left(\frac{n-k+\frac{1}{2}}{n}\pi\right)$$

Let's take these out for a spin using the code `approxtest.m`. Use $z=\{1,2,20\}$ with the number of nodes being $\{5,10,20\}$. The moral of the story is that

- You can't approximate everything arbitrarily well. You need to have some idea of the smoothness of your function or you will loose detail.
- Check approximations by computing some points off the interpolation grid and checking the error.

3.3 Continuous State Dynamic Programming

Let's return to our deterministic growth problem one final time:

$$V(k) = \max_{k'} u(k^\alpha + (1-\delta)k - k') + \beta V(k')$$

We wish to solve for V and avoid discretizing the state space, K. Instead, let's define our approximation of V as:

$$V(k) \approx \tilde{V}(k; c) = \sum_{j=1}^{n} c_j \phi_j(k)$$

Where,

- $\{\phi\}$ is a family of basis functions (we'll use Chebyshev polynomials).
- c is the n-dimensional parameter that defines our approximation. Given an initial c^0 we have an initial guess of V (complete with derivatives, if we want them!).

To perform value funciton iteration, we need to solve the agent's optimization problem, but only on a grid of n nodes (for us, the Chebyshev nodes). E.g., solve:

$$V_j = \max_{k'} u(k_j^\alpha + (1-\delta)k_j - k') + \beta \tilde{V}(k', c^{r-1}) \tag{3.1}$$

The argmax does *not* need to be a basis node, it can be anything in R^+. In fact, we could use the first order condition to solve this problem:

$$-u'(k_j^\alpha + (1-\delta)k_j - k') + \beta \tilde{V}'(k', c^{r-1}) = 0$$

Where $\tilde{V}'$ is simple because $\tilde{V}$ is a polynomial:

$$\tilde{V}'(k', c) = \sum_{j=1}^{N} c_j \phi_j'(k')$$

For our simple problem, we can just use `fmincon` to solve this.

Once we have found V_j at all interpolation nodes, these are our "data" to update our guess of V.

Value Function Iteration Loop:

1. Given c^{r-1} for each iteration node k_j
   - Evaluate V_j by solving (3.1).
2. Calculate new c using,
   $$c^r = \Phi^{-1} V$$
3. If $||c^r - c^{r-1}|| < \varepsilon$, stop, otherwise go to 1.

Put it into action with the file `growmodel_co.m`.

3.4 Interpolating Functions of Multiple Dimensions

In principle, it is easy to extend what we have done so far to higher dimensions, just use the tensor product basis:

If A is a basis for x and B is a basis for y, the tensor product basis for (x, y) is,

$$A \otimes B = \{\phi(x)\psi(y) | \phi \in A, \psi \in B\}$$

So if A has n functions and m has n in the end we have $m \times n$ functions in our tensor product basis. Our approximation becomes:

$$\hat{f}(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} \phi(x) \psi(y)$$

This will work great for 2, maybe 3 dimensions, the obvious problem is that the number of nodes is growing exponentially in the number of dimensions (i.e., the curse of dimensionality).

There are some ways of dealing with this:

- Sparse grids are formulated to eliminate some of these points and "re-weight" remaining points.
- Ken Judd has been working on a stochastic approximation method "Gerneralized Stochastic Simulation Algorithm" (GSSA).
- This is an active area, but for most applied work, we realize the problem and have been attempting to keep the state space as small as possible to answer the question we are interested in.