Interpolation

Garrett A. Anstreicher

Thursday 11th June, 2020

Motivation

Think back to value function iteration

- We expressed the problem as

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

- This is a bit misleading: on computer, only looped over possible next-period choices that were actually in the discretized capital grid:

$$V(k) = \max_{k' \in D^k} u(c) + \beta V(k')$$
$$k' = k^\theta + (1-\delta)k - c$$

- This raises some issues
  - What if best option is in between points?
  - What to do if random shocks?
  - What does an agent do at in-between capital states?

What we want

- Take an arbitrary function evaluated at a discrete set of points.
- Create an approximation function with continuous domain.
- This process is broadly called **interpolation**
- Several useful things about this
  - More flexibility in choices when forming policy functions
  - Can reduce state space considerably
- Note that none of this is necessary if the function has a closed form.
  - $V(k)$ from last slide does not!

Two common types of interpolation

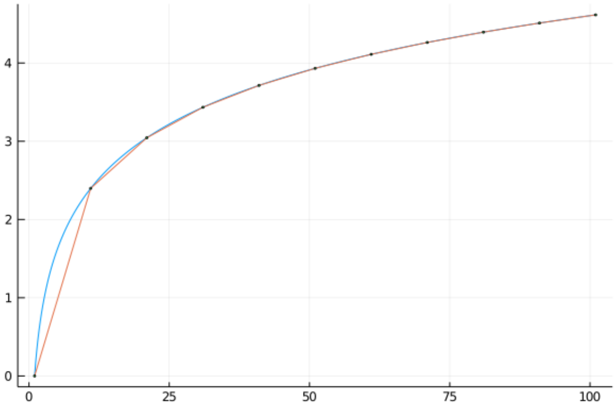
- Polynomial interpolation
  - Fit a polynomial of some degree to the points and then use the polynomial going forward
  - Problem: Runge's phenomenon. Odd behavior at tails.
- Linear/spline interpolation
  - Connect-the-dots
  - Can connect dots with straight line or be more fancy

Linear Interpolation

- Used the most frequently. Simple idea.
- Start with discretized domain $[x_1, ...x_n]$ and range $[f(x_1), ...f(x_n)]$.
- Linear interpolation is a function $L(x)$ such that:
  - $L(x_i) = f(x_i) \ \forall i$
  - For any $x \in [x_i, x_{i+1}]$:

$$L(x) = f(x_i) + (x - x_i) \cdot \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

Visualization



What if function is multivariate?

- No problem! You can interpolate over as many dimensions as you want.
- 2-d case: **bivariate interpolation**
- If $x \in [x_1, x_2]$ and $y \in [y_1, y_2]$, then

$$B(x,y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \cdot \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \begin{bmatrix} f(x_1, y_1) & f(x_1, y_2) \\ f(x_2, y_1) & f(x_2, y_2) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}$$

- **Caution:** interpolating in many (>3) dimensions can be very computationally expensive.

Cubic Spline Interpolation

- More complicated. Now fitting cubic function between points (can do quadratic, too).
- Same discretized domain and range. Now construct a function $S(x)$, such that:
  - $S(x_i) = f(x_i) \ \forall i$.
  - In any interval $[x_i, x_{i+1}]$, we have:

$$S_i(x) = a_i + b_i x + c_i x^2 + d_i x^3$$

- Note that the function coefficients vary based on what interval we're in.
- Since there are N points, there are $(N-1)$ intervals, and we thus have $4 \cdot (N-1)$ coefficients to solve for.

Solving for the coefficients

- We get $2(N-1)$ equations from endpoint conditions: $S_i(x_i) = f(x_i)$ AND $S_{i-1}(x_i) = f(x_i)$
- Continuity of derivatives on the interior nodes gives $(N-2)$ equations: $S'_{i-1}(x_i) = S'_i(x_i)$
- Continuity of second derivative also gives $(N-2)$ equations: $S''_{i-1}(x_i) = S''_i(x_i)$

Solving for the coefficients

- So now we have $2(N-1)+2(N-2)=4N-6$ equations. Note that this is less than the number of unknowns, which is $4(N-1)$.
- We need two more conditions.
- Typical way around this is to assume that first derivatives at endpoints are equal to rise-over-run from nearest interior points.

Visualization