julia> Lecture Four: Interpolation

Course: Computational Bootcamp
Name: John Higgins
Date: June 26, 2024

julia> Today's goals

1. Understand the basics of interpolation techniques
2. Understand how they can be applied to solve dynamic programming problems

julia> Motivation

> Remember the optimal capital accumulation problem we solved last week:

$$V(k) = \max_{c,k'} \left\{ \log(c) + \beta V(k') \right\}$$

$$c + k' = k^{\alpha} + (1 - \delta)k$$

> We forced capital to be in a discrete set:

$$D_k = \{k_1, \ldots, k_N\}$$

> And solved the value function assuming we could only choose values within this discrete set:

$$V(k) = \max_{k' \in D_k} \left\{ u(c) + \beta V(k') \right\}$$

julia> Problems

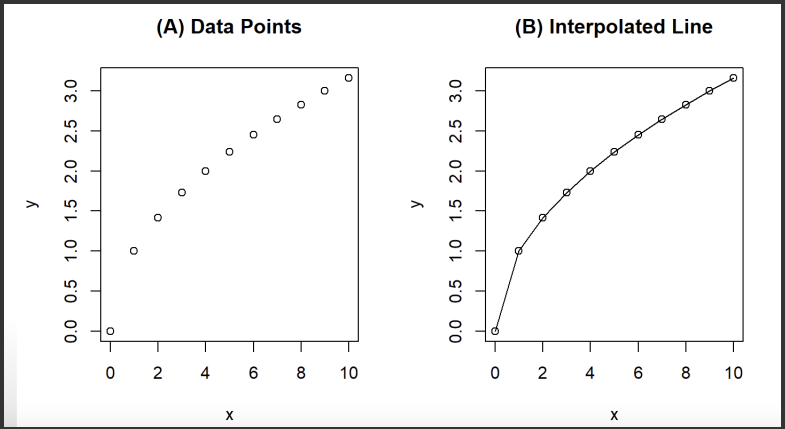
> This is a good way to get started, but:

1. It forces you to do optimization via grid search
2. Might need a lot of grid points to get a good approximation
3. The optimal decision might be between or outside the grid
4. Don’t know what agent does when between grid points

julia> Interpolation

> Interpolation: the approximation of a function within a set of known points

> In plain English: connect-the-dots for PhD Economists



julia> Types of interpolation

- There are many ways to connect the dots because you can do whatever you want in between them
- Common ways:
  - Polynomial Interpolation: Fit degree N polynomial to the data
    - Motivation: any continuous function on compact support can be approximated by polynomials (Stone-Weierstrass)
    - Problem 1: Data points are not always on the fitted line
    - Problem 2: Strange things can happen at the tails, especially as N increases (Runge's phenomenon)
  - Linear/Spline Interpolation: Draw lines between points
    - Linear: Assume that the line is straight.
    - Spline: Assume the line is a polynomial of degree N

julia> Linear interpolation

> Discrete domain:

$$\{x_1, x_2, \ldots, x_N\}$$

> Discrete range:

$$\{f(x_1), f(x_2), \ldots, f(x_N)\}$$

> Linear interpolation $L(x)$: for $x \in [x_i, x_{i+1}]$,

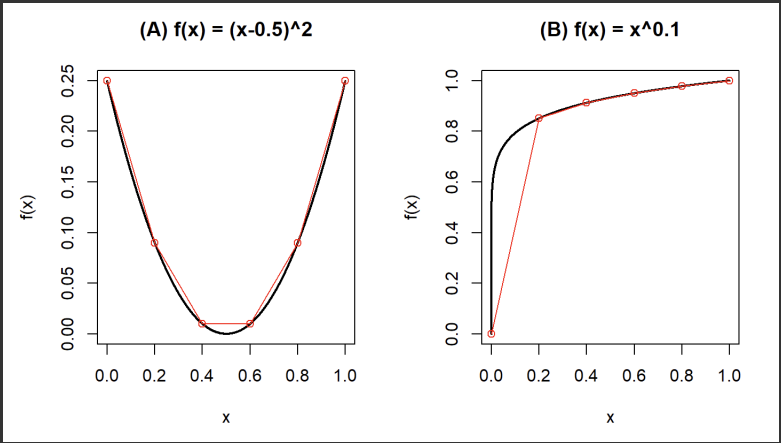
$$L(x) = f(x_i) + (x - x_i)\frac{f(x_i + 1) - f(x_i)}{x_{i+1} - x_i}$$

> Intuition: weighted average of function values based on distance

> Problems:

1. Derivative can change drastically at grid points.
2. Derivative is constant between grid points

julia> Linear interpolation examples



[~]$ _

julia> Cubic spline interpolation

> Fit cubic polynomial between all points
> N points $\{(x_i, f(x_i))\}_{i=1}^{N}$
> When $x \in [x_i, x_{i+1}]$:

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

> This adds some curvature to the approximation (not always a good thing!)
> N points, so there are $N-1$ intervals and $4(N-1)$ coefficients

julia> Solving for spline coefficients

> $2(N-1)$ end point conditions:

$$S_i(x) = f(x_i) \quad \& \quad S_{i-1} = f(x_i)$$

> $2(N-2)$ continuity of 1st and 2nd derivative conditions:

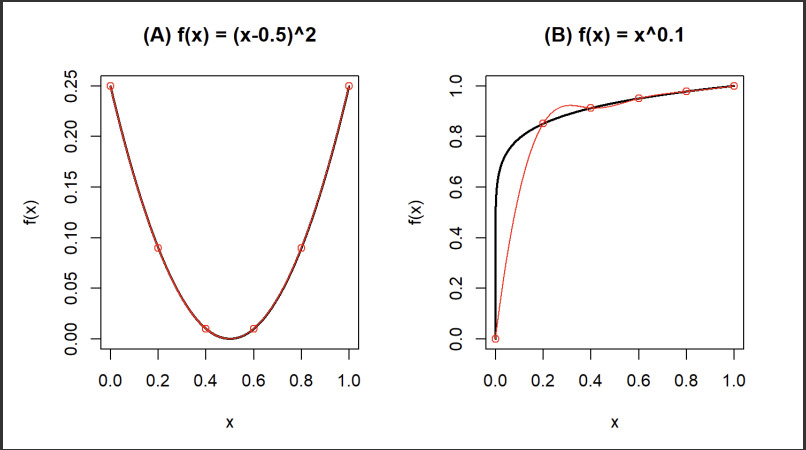
$$S'_{i-1}(x_i) = S'_i(x_i) \quad \& \quad S''_{i-1}(x_i) = S''_i(x_i)$$

> Two final equations: Assume derivatives at endpoints are equal to rise-over-run from nearest endpoints

> Benefits: Cubic Interpolation is smooth with continuous derivatives

> This can cause problems sometimes

julia> Cubic spline interpolation examples



julia> Multivariate interpolation

> These methods can be extended to multivariate case using bivariate linear interpolation

> Discrete domain:

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$

> Discrete range:

$$f(x_1, y_1), f(x_2, y_2), \ldots, f(x_N, y_N)$$

> Bivariate Linear Interpolation $B(x, y)$: for $(x, y) \in [x_i, x_{i+1}] \times [y_i, y_{i+1}]$

$$B(x,y) = \frac{1}{(x_{i+1} - x_i)(y_{i+1} - y_i)} \begin{bmatrix} x_{i+1} - x & x - x_i \end{bmatrix} \times \begin{bmatrix} f(x_i, y_i) & f(x_i, y_{i+1} \\ f(x_{i+1}, y_i) & f(x_{i+1}, y_{i+1}) \end{bmatrix} \begin{bmatrix} y_{i+1} - y \\ y - y_i \end{bmatrix}$$