Function Approximation

Judd Chapter 6

David Childers (thanks to Y. Kryukov, K. Judd, and U. Doraszelski)

CMU, Tepper School of Business

Mar 27-29, 2023

The Plan

- Motivation
- Local approximations; Taylor, Pade, logs
- Interpolation
  - Polynomials
  - Splines
- Next Class: More Global Methods
  - Projection and Regression
  - Orthogonal basis functions
  - Multi-dimensional approaches

Motivation, example 1

Bellman's functional equation:

$$V(k) = \max_{k' \geq 0} \pi(k) + \beta V(k')$$

- Want to find $V(\cdot)$, e.g. via fixed-point iteration:

$$V^{k+1}(k) = \max_{k' \geq 0} \pi(k) + \beta V^{k}(k') \qquad (*)$$

- To find max, we need $V^k(k')$ or its derivative, at any k'
  - We need a differentiable approximation $\hat{V}(\cdot)$
- Don't want to compute exact $V^k(k)$ for too many values of k
  - Instead, use them to construct the approximation

Motivation, example 2

- Have utility function $u(x_1, x_2)$, want Demand for good 1
- Need to find $D_1(p_1) = D(p_1, \bar{p}_2)$, where

$$\begin{aligned} D(p) &= \arg\max u(x_1, x_2) \qquad (**) \\ \text{subject to} &: \quad p_1 x_1 + \bar{p}_2 x_2 \leq B \end{aligned}$$

- Will need to evalute $D_1(p_1)$ at many points,
  e.g. when integrating for CS, or solving monopolist's problem
- Want to solve (**) for a few points, and
  - Construct an accurate approximation to $D_1(p_1)$
  - That is easy to integrate or differentiate analytically
    (i.e. a polynomial)

Approximation stages

1. Choose approach (family $\mathcal{F}_m$ of approximating functions, eg polynomials of degree m)
2. Obtain "data" for estimation
   - Choose n *functionals*: $f_i(y()) \in \mathbb{R}$ to evaluate
   - E.g., choose node(s) x_i and obtain $y_i \equiv y\left(x_i\right)$
   - Data may also contain derivatives $y'\left(x_i\right)$ or integrals $\int f_i(x)y(x)dx$
3. Map data to (representation of) $\hat{y}(.) \in \mathcal{F}_m$ e.g. coefficients
4. Evaluation: plug in any x, get back $\hat{y}\left(x\right)$

- "Approximation" could refer to any of these stages
- Each stage is repeated more frequently than previous ones

Local approximation: Taylor series

- Degree n Taylor approximation to $f(x)$ around x_0:

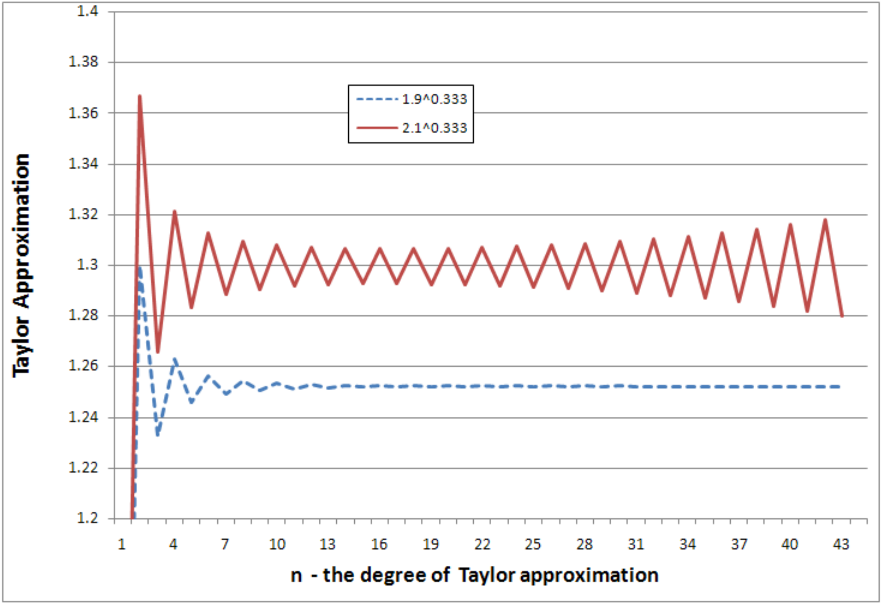
$$
\begin{aligned}
t(x) &= f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2 \\
&\quad + \ldots + \frac{1}{n!} f^{(n)}(x_0)(x - x_0)^n
\end{aligned}
$$

- Error grows rapidly with $|x_0 - x|$;
  declines with n within the *convergence radius* r
- Theorem 6.1.2: $r \leq$ distance to nearest singularity of f
- Singularity: x such that $f^{(n)}(x) = \infty$

Example:

- $f(x) = x^{\frac{1}{3}}$ has a singularity at $x = 0$.
- $\Rightarrow$ Taylor series expansion around $x_0 = 1$
  breaks down outside $[0, 2]$

Taylor approximation around $x = 1$



Pade approximation

- *Taylor*: polynomial of degree n, $t^{(i)}(x_0) = f^{(i)}(x_0)$, $i \leq n$
- **Pade**: rational function:

$$r(x) = p(x)/q(x)$$

  $p(x)$ – polynomial of degree n; $q(x)$ – of degree m
- Coeff's of p and q – determined from a system of linear equations:

$$\frac{d^k}{dx^k}(p - f * q)(x_0) = 0, \quad k = 0, \ldots, m + n$$

- More accurate than Taylor, not affected by singularities
- Read Judd on simplifying the implementation

Log-Linearization: one example

Linear approximation to % deviations (from steady state)

- Take a production function: $y_t = z_t k_t^{\alpha}$.
  - Take logs: $\ln y_t = \ln z_t + \alpha \ln k_t$
  - Now exactly linear
- To interpret: compare to linearization in levels
- We know the steady state: (y^*, z^*, k^*)
  - Taylor around it: $\ln y_t \approx \ln y^* + (y_t - y^*)/y^*$
  - Same for $\ln z_t$, $\ln k_t$
- Plug these into log of production function.
  Use $\ln y^* = \ln z^* + \alpha \ln k^*$ to derive:

$$\frac{y_t - y^*}{y^*} \approx \frac{z_t - z^*}{z^*} + \alpha \frac{k_t - k^*}{k^*}$$

Log-Linearization: another example

- Implicit function: $y(x)$ defined by $f(y,x)=0$
- Implicit function theorem:

$$\frac{dy}{dx} = -\frac{\partial f/\partial x}{\partial f/\partial y} = -\frac{f_x}{f_y}$$

- Use the fact that $d\ln x = dx/x$:

$$d\ln y = -\frac{x}{y}\frac{f_x}{f_y} * d\ln x$$

- Approximation:

$$\ln y - \ln y_0 \approx -\frac{x_0 f_x(y_0,x_0)}{y_0 f_y(y_0,x_0)}(\ln x - \ln x_0)$$

Example of local methods

Exercise 6.1: $f(x) = \left(x^{\frac{1}{2}} + 1\right)^{\frac{2}{3}}$, $x_0 = 1$.

- Degree 1 and 2 Taylor series approximation:

$$\begin{aligned} t_1(x) &= 2^{\frac{2}{3}} \left(1 + \tfrac{1}{6}(x-1)\right), \\ t_2(x) &= 2^{\frac{2}{3}} \left(1 + \tfrac{1}{6}(x-1) - \tfrac{7}{144}(x-1)^2\right), \end{aligned}$$

- Padé approximation (1,1):

$$r(x) = 2^{\frac{2}{3}} \frac{24 + 11(x-1)}{24 + 7(x-1)}.$$

- Log-linear approximation ($\ln x_0 = 0$):

$$l_1(x) = 2^{\frac{2}{3}} \left(1 + \tfrac{1}{6}\ln(x)\right)$$

Local methods: Pros and Cons

- Cons: Obvious
  - Limited radius of convergence
  - Moderate accuracy within radius
  - Need derivatives, can't handle kinks or discontinuities
- Pros:
  - Linear functional forms allow fast calculations by linear algebra even in extremely high dimensions
  - Can leverage fast first order solutions to solve higher order
  - Huge speed gains, good automated software make this first choice
  - At worst, use for rapid prototyping and good initial guess for global methods

Global methods: introduction

Polynomial approximation: $f(x) \approx \sum_{j=0}^{m} a_j p_j(x) = \hat{f}(x;a)$

1. Pick n nodes x_i, compute $f(x_i)$
2. Pick coefficients a to minimize $\left| f(x_i) - \hat{f}(x_i) \right|$'s

- $n = m + 1$ – interpolation, can fit $f(x_i)$'s exactly
- $n > m + 1$ – regression, typically least squares
    - Unlike Econometrics, we can pick x_i's and $p_j(\cdot)$'s

Splines: piece-wise polynomials (of low degree)

- split x interval into $\inf x = x_0 < x_1 < ... < x_m = \sup x$
- On each $[x_j, x_{j+1}]$, $\hat{f}(x) = \hat{f}_j(x)$ is a low-order polynomial.
- Impose continuity: $\hat{f}_{j-1}(x_j) = \hat{f}_j(x_j) = f(x_j)$, smoothness: $\hat{f}'_{j-1}(x_j) = \hat{f}'_j(x_j)$, etc.

Polynomial approximation by interpolation

- Have 'data': $x_i, y_i = f(x_i)$, $i = 1, \ldots, n$
- Find a polynomial of degree $n-1$ that satisfies

$$p(x_i) = y_i, \quad i = 1, \ldots, n.$$

- n conditions $\Rightarrow$ n linear equations pinning down the n coefficients of $p(\cdot)$:

$$a_0 + a_1 x_i + a_2 x_i^2 + \ldots + a_{n-1} x_i^{n-1} = y_i, i = 1, \ldots, n$$

- System of linear equations $Va = y$
- $V =$ Vandermonde matrix, often has large condition number
- Lots of computation for new y_i's (solving $Va = y$); few computations to evaluate $p(x)$ for an arbitrary x
- Precompute QR if repeatedly computing coefficients w/ new data at same points
  - $O(n^3)$ overhead for $O(n^2)$ per step

Direct approach: Lagrange interpolation

- If coefficients not needed, can avoid computing them completely
- Explicit solution to $p(x_i) = y_i$ equations:

$$p(x) = \sum_{i=1}^{n} y_i l_i(x),$$

where

$$l_i(x) = \prod\nolimits_{j \neq i} \frac{x - x_j}{x_i - x_j}.$$

  - Note that $l_i(x_i) = 1$, $l_i(x_j) = 0$ if $i \neq j$.
- In practice, do not compute directly: slow and unstable

Better implementation: Barycentric formula

- Use *Barycentric Formula* to find polynomial interpolation
- Set $p(x_i) = y_i$ and for all other x

$$p(x) = \sum_{i=1}^{n} \frac{\lambda_i y_i}{x - x_i} / \sum_{i=1}^{n} \frac{\lambda_i}{x - x_i}$$

where

$$\lambda_i(x) = \frac{1}{\prod_{j \neq i}(x_i - x_j)}.$$

- $O(n^2)$ to get weights λ_i, then $O(n)$ per evaluation
    - Same order as if polynomial known even w/ new data y_i
- Is polynomial interpolation a good idea? Depends on choice of x_i

Problems with polynomial interpolation

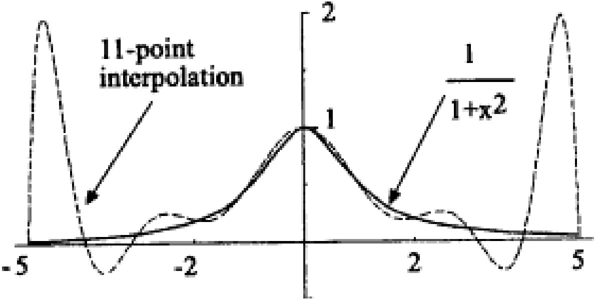


Figure 6.6
Nonconvergence of interpolation example

- Runge Phenomenon: interpolation at evenly spaced points
- Interpolation may diverge even if true function $\infty-$differentiable
- Solution: use stable point set, non-polynomial functions, or projection instead of interpolation

Possible improvement: Hermite interpolation

- Have data: x_i, $y_i = f(x_i)$, $y_i' = f'(x_i)$, $i = 1, \ldots, n$
- Find a polynomial of degree $2n - 1$ that satisfies

$$p(x_i) = y_i, \quad p'(x_i) = y_i' \quad i = 1, \ldots, n.$$

$2n$ conditions $\Rightarrow$ $2n$ linear equations to pin down the $2n$ coefficients of p.

- Explicit solution:

$$p(x) = \sum_{i=1}^{n} y_i h_i(x) + \sum_{i=1}^{n} y_i' \tilde{h}_i(x),$$

where

$$\begin{aligned} \tilde{h}_i(x) &= (x - x_i)\left[l_i(x)\right]^2, \\ h_i(x) &= \left(1 - 2l_i'(x_i)(x - x_i)\right)\left[l_i(x)\right]^2. \end{aligned}$$

Note $l_i'(x_i) = \sum_{j \neq i} \frac{1}{x_i - x_j}$.

Major improvement: Good point sets

- Optimal point set depends on true and approximating functions, desired convergence class
- For polynomials on $[-1, 1]$, smooth functions, uniform approximation, near best are Chebyshev points

$$x_i = \cos(\frac{i\pi}{n}) \ \{i \in 0 \dots n\}$$

- Thm: Let $f(x)$ have k^{th} derivative with total variation V and let $\hat{f}_n(x)$ be interpolation using $n+1$ Chebyshev points

$$\left\| \hat{f}_n(x) - f(x) \right\|_\infty \leq \frac{4V}{\pi k (n-k)^k}$$

- Many other desirable computational properties: next class

Alternative for general point sets: Splines

A function $s(x)$ on $[a, b]$ is a spline if:

- There is a grid of points (nodes or knots) $a = x_0 < x_1 < \ldots < x_{n-1} < x_n = b$, ...
- ... such that $s(x)$ is a polynomial of degree k on each subinterval $[x_{i-1}, x_i]$, $i = 1, 2, \ldots, n$, ...
- ... and $s(x) \in \mathbf{C}^{k-1}$ (continuously differentiable to $k - 1$ order)
- Order of the spline can be defined as k or $k + 1$

- Linear spline: Piecewise linear, "connecting the dots."
- Cubic spline: $s(x) = a_i + b_i x + c_i x^2 + d_i x^3$.

- Low cost of computation: efficient sparse matrix algorithms
- Rapid convergence when $|x_{i-1} - x_i| \to 0$ rate $= k - 1$ if $f \in \mathbf{C}^{k-1}$
- Good for functions that are not $\mathbf{C}^\infty$

Example: cubic spline

$s(x) = a_i + b_i x + c_i x^2 + d_i x^3$ on $[x_{i-1}, x_i]$.

- $2n$ interpolation conditions:

$$\begin{aligned} y_i &= a_i + b_i x_i + c_i x_i^2 + d_i x_i^3, \quad i = 1, \ldots, n, \\ y_i &= a_{i+1} + b_{i+1} x_i + c_{i+1} x_i^2 + d_{i+1} x_i^3, \quad i = 0, \ldots, n-1, \end{aligned}$$

- $2n - 2$ derivative conditions:

$$\begin{aligned} b_i + 2c_i x_i + 3d_i x_i^2 &= b_{i+1} + 2c_{i+1} x_i + 3d_{i+1} x_i^2, i = 1, \ldots, n-1 \\ 2c_i + 6d_i x_i &= 2c_{i+1} + 6d_{i+1} x_i, \quad i = 1, \ldots, n-1. \end{aligned}$$

- Need 2 end conditions. E.g., the secant Hermite spline:

$$s'(x_0) = \frac{y_1 - y_0}{x_1 - x_0}, \quad s'(x_n) = \frac{y_n - y_{n-1}}{x_n - x_{n-1}},$$

- $4n$ linear equations identify the $4n$ coefficients of s.

More splines

B-splines (de Boor (1978)): good for deriving analytical results about other splines

- A basis of orthogonal functions
- Each function is non-zero over two adjacent intervals
- Recursive formulation $\Rightarrow$ increasing smoothness

Shape-preserving splines:

- Concave data might lead to non-concave spline
- Schumaker procedure
- Intermediate nodes $z_i \in [x_{i-1}, x_i]$, data on $f'(x_i)$

Recap of approximations

- Want $f\left(x\right)$ for many different x's
  - $f\left(\cdot\right)$ takes a long time to evaluate
  - $\Rightarrow$ use approximation $\hat{f}\left(x\right) = \sum_{k=0}^{m} a_k p_k(x)$
- Want to find a_k with as little data as possible:
  - Local around x_0: $f\left(x_0\right)$, $f'\left(x_0\right)$, $f''\left(x_0\right)$, ..., $f^{(n)}\left(x_0\right)$
  - Global: $f\left(x_1\right)$, $f\left(x_2\right)$, ..., $f\left(x_n\right)$
- Similar to prediction in Econometrics
  - But econometrics uses observational data
  - Here, we can "design the experiment"
- Smart choice of polynomials $p_k(\cdot)$ and nodes x_i.

Basis Function Methods

- Represent function by finite set of coefficients on known set of functions $\phi_k()$
- Approximation: $p(x) = \sum_{k=0}^{n} a_k \phi_k(x)$
- **Mean Square** (L^2) approximation:

$$\min_a \int [f(x) - p(x)]^2 w(x) dx \Leftrightarrow \min_a \|f - p\|_2$$

- **Uniform** (L^∞) approximation:

$$\min_a \sup_x |f(x) - p(x)| \Leftrightarrow \min_a \|f - p\|_\infty$$

- Usually don't target optimal uniform approximation directly
  For many classes (Chebyshev, B-spline, Wavelet, Fourier), good L^2 approx implies good L^∞
- **Derivatives** may not converge

Orthogonal bases

- Define ($w(x)$ weighted) inner product:

$$\langle f, g \rangle = \int f(x)g(x)w(x)dx.$$

- The family of functions $\{\phi_k\}$ is **mutually orthogonal** iff $\langle \phi_k, \phi_m \rangle = 0$ for all $k \neq m$.
    - They *span a subspace* within the space of functions
- Optimal L^2 approximation has simple formula when ϕ_k *orthogonal*
    - $a_k^* = \frac{< f, \phi_k >}{< \phi_k, \phi_k >}$
    - Can calculate separately by direct computation of integral
- Orthogonality also leads to better conditioning when regression approach used

Why orthonormality?

- Complete orthonormal basis gives mean square representation
- *Parseval's theorem*:
  - Let $\|f\|_2^2 := \int f(x)^2 w(x)dx < \infty$ and $\{\phi_k\}_{k=1}^{\infty}$ be a complete o.n.b. Then

$$\left\| \sum_{k=1}^{N} \langle f, \phi_k \rangle \phi_k - f \right\|_2^2 = \sum_{k=N+1}^{\infty} a_k^2$$

- Allows representation of continuum by countable family
  - and good, *quantifiable* approximation by finite family
- L^2 good enough for some things
  - Integrals, some functional equations, *not* pointwise convergence
- Compare Weierstrass's theorem: Let f continuous, then $\inf_{p_N} \sup_x |f(x) - p_N(x)| \to 0$
- Fine, but no speed results, sequence may depend on (unknown) f

From convergence to rates

- When does projection representation ensure pointwise convergence?
- Function must live in strict subset of L^2
- Many many classes exist, usually "smoothness conditions"
  - $\mathcal{C}^k$ k-times continuously differentiable,
  - Sobolev: $W^{p,k} := \left\{ f : \left\| \frac{d^k}{dx} f(x) \right\|_p < \infty \right\}$
  - Holder: $\Lambda^\alpha := \left\{ f : \sup_{x,y} \left| \frac{d^{\lfloor a \rfloor}}{dx} f(y) - \frac{d^{\lfloor a \rfloor}}{dx} f(x) \right| \leq C d(x,y)^{a - \lfloor a \rfloor} \right\}$
  - See also: Besov, Bounded Variation, Reproducing Kernel Hilbert Spaces, etc
- Each class admits rates in some norms for some basis set
- Technique: demonstrate class membership by analytical methods (eg map preserves derivatives or integrals, etc)
- Then use suitable basis given class knowledge

Orthogonal families

Family	$w(x)$	**Domain**	**definition**
Legendre	1	$[-1,1]$	$P_n(x) = \frac{(-1)^n}{2^n n!} \frac{d^n}{dx^n} \left((1-x^2)^n\right)$
Chebyshev	$(1-x^2)^{-\frac{1}{2}}$	$[-1,1]$	$T_n(x) = \cos\left(n \cos^{-1}(x)\right)$
Laguerre	e^{-x}	$[0,\infty)$	$L_n(x) = \frac{e^x}{n!} \frac{d^n}{x^n} \left(x^n e^{-x}\right)$
Hermite	e^{-x^2}	$(-\infty,\infty)$	$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} \left(e^{-x^2}\right)$
Fourier	1	$[-\pi,\pi]$	$\frac{1}{2}; \cos(nx); \sin(mx)$
Wavelet	1	$[-1,1]$	No closed form

- Look for domain and shape that fits your application
- Chebyshev is a common choice in Economics
- Fourier is used for periodic functions, rare in Econon
- Useful polynomial families have recurrence formulas:
  - $\phi_0(x) = 1, \qquad \phi_1(x) = x$
  - $\phi_{k+1}(x) = \left[c_{k+1} x + d_{k+1}\right] \phi_k(x) + e_{k+1} \phi_{k-1}(x)$

Details: Legendre

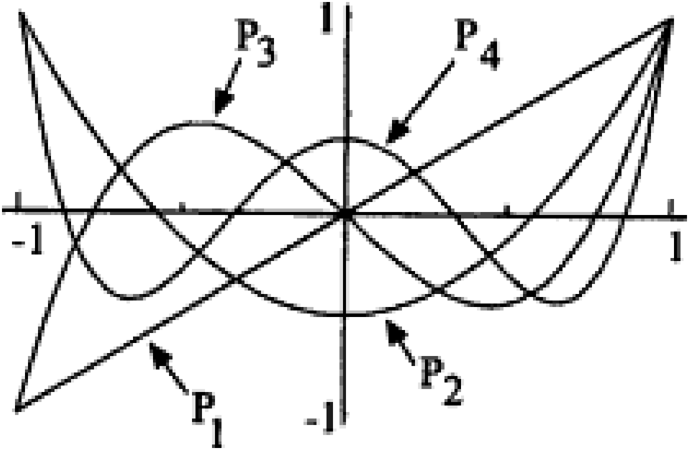


Figure 6.2
Legendre polynomials

Details: Chebyshev

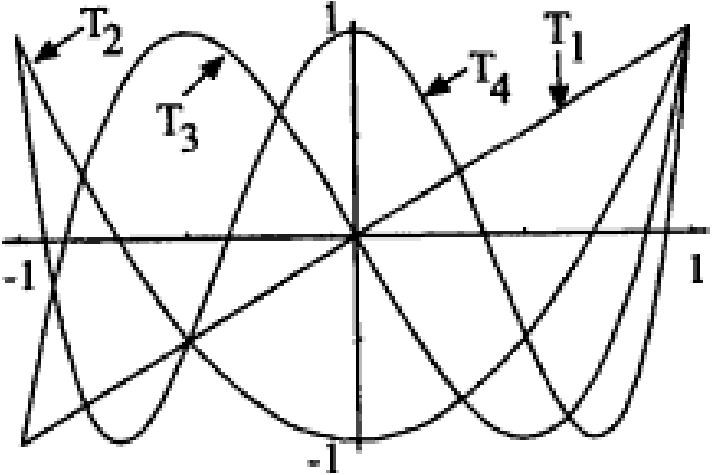


Figure 6.1
Chebyshev polynomials

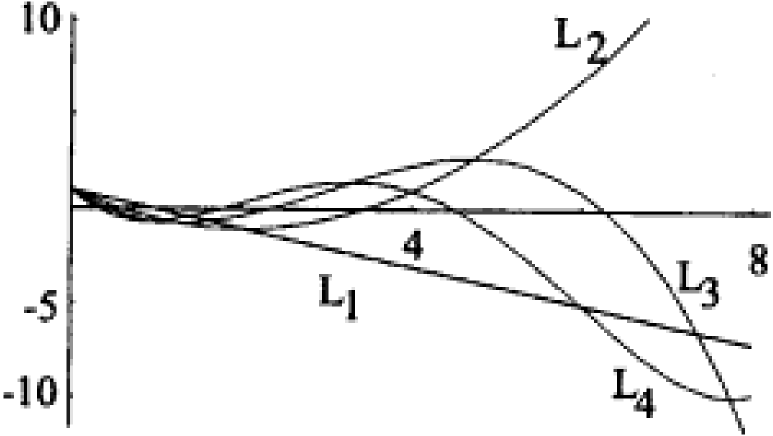


Figure 6.3
Laguerre polynomials

Details: Hermite

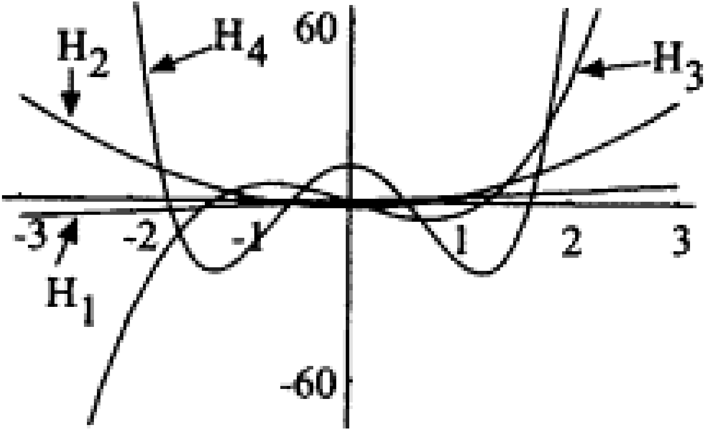


Figure 6.4
Hermite polynomials

Regression Approach to Finding Coefficients

1. **Set-up** (at beginning of code):
    1. Choose functions $\phi(\cdot)$, order n, and # of nodes $m \geq n+1$
    2. Select nodes z_k in the function's support, $k = 1, ..., m$: polynomial zeros, or just a uniform grid
    3. Map each z_k into x_k in the support of the target function
    4. Evaluate function $\phi_j(z_k)$, $j = 0, ..., n$;

       store them as a matrix $X : X_{kj} = \left\{\phi_j(z_k)\right\}$
2. **Estimation** of coefficients $a = [a_0, ..., a_m]$:
    1. Evaluate $y_k = f(x_k)$; form them into vector Y
    2. If $m = n+1$, solve $Xa = Y$
    3. If $m > n+1$, solve $(X'X)\, a = X'Y$
3. **Evaluation** of $\hat{f}(\bar{x})$ an for an arbitrary $\bar{x}$
    1. Map $\bar{x}$ into $\bar{z}$ (i.e. into the support of ϕ)
    2. Compute $\phi_j(\bar{z})$'s; recursive formula will save time
    3. Compute $\hat{f}(\bar{x}) = \sum_{j=0}^{n} a_j \phi_j(\bar{z})$

Properties of regression approach

- When $m = n$, this is just interpolation:
  accurate for Chebyshev on Chebyshev points (or Fourier on uniform grid), not otherwise
- In both cases, interpolation system solved in $n \log(n)$ time by *Fast Fourier Transform*
- Choose $n >> m$ for general case to converge to projection coefficients
- Accuracy of order n projection depends on f and $\{\phi\}$
- For polynomials: Jackson and Bernstein inequalities:

  $$\inf_{p_n} |f - p_n| \leq Cn^{-(k+a)}$$

  holds iff f has k derivatives which are a-Holder-continuous
- Use theory to derive properties, pick suitable class

Changing (bounded) domain

Linear change of variable preserves orthogonality:

- $[a, b]$ to $[-1, 1]$: $z = -1 + 2\frac{x-a}{b-a}$
- $[a, +\infty]$ to $[0, +\infty]$: $\frac{1}{\lambda}(x - a)$
  - λ helps match the area of curvature of the polynomial

Other useful transformations:

- $(-\infty, +\infty)$ to $(0, +\infty)$: e^x
- $(0, +\infty)$ to $(-\infty, +\infty)$: $\ln x$
- $(-\infty, +\infty)$ to $[0, 1]$: $1/(e^{-x} + 1)$ = the logistic function
- $[0, 1]$ to $(-\infty, +\infty)$: inverted logistic function

Changing unbounded domain

- Laguerre: $[0, \infty)$. Hermite: $(-\infty, +\infty)$
- Math will work without change of domain
- But polynomials have curvature and extrema concentrated in a neighborhood of 0
  - E.g. Laguerre deg. 4 no extrema outside $[0, 10]$
- If $f(x)$ has most curvature happening on $[0, \bar{X}]$, it might make sense to rescale: $z = x\,(10/\bar{X})$
- You can experiment with rescaling to obtain the best fit

Nonlinear regression approach

- Nothing in principle requires using sum of basis functions
- Can choose nonlinear class, find parameters by nonlinear least squares
- Adaptive regression splines: choose knot points and coefficients
- Neural networks: repeatedly compose linear and nonlinear maps
  - Single *neuron* j is $F_j(\sum_{k=1}^{n} x_{jk}\beta_{jk})$ for pointwise nonlinear function like $F(x) = \max(x, 0)$ or logit(x) with inputs x_{jk} and weights β_{jk}
  - A set of neurons form a *layer*, and are used as inputs to another set of neurons
- Optimization problem harder, so choose only if nonlinear class gives much better approx of true function
  - Adaptive splines improve rates for Bounded Variation functions
  - Neural nets perform well for high-dimensional, irregular functions: space of images, sounds, text

Adaptive methods

1. Start with initial grid,
2. Use one of above methods to approximate
3. Pick new points, compare approximation and truth
4. Increase order of approximation based on errors

- Especially useful if easy to refine: e.g. splines, wavelets
- Alternative: penalized or thresholded approximation
  - Find coefficients, then remove small ones
  - or use procedure that gives only small set of coefs (eg LASSO)
- Result: much smaller set of coefficients
  - Useful if next step has costs depending on number
- Accuracy cost low if true function *sparse* in basis
  - Only a few (knots, wavelets etc) matter:
  - True if function has small set of spikes/inflections
  - E.g. Models with hard constraints can give sharp breaks

Wavelets

- Bespoke basis functions designed for fast optimal approximation
- Start with pair $\phi(x)$ $\psi(x)$ father and mother wavelet (or scaling function and wavelet)
- Scaling function self-similar: $\phi(x) = \sum_{k=1}^{n} a_k \phi(2x - k)$
- Wavelet $\psi(x) = \sum_{k=1}^{m} b_k \phi(2x - k)$
- $\{\phi_{j,k}(x) = 2^{-j/2}\phi(2^{-j}x - k)\}$ $\{\psi_{j,k}(x) = 2^{-j/2}\psi(2^{-j}x - k)\}$ are basis collection
- n coefs calculated recursively in $O(n)$ time by cascade algorithm
- Can choose so compactly supported, orthogonal, approximate smooth/nonsmooth functions
- In adaptive methods, use only large coefs $\rightarrow$ good approx if function has nonsmooth area, like edge
- Many families available for different applications

Multi-dimensional approximation

$F(x) : \mathbb{R}^m \to \mathbb{R}$

- *Lagrange interpolation* – requires careful choice of nodes
    - Otherwise, coefficient equations can become singular
- *Tensor products* of orth. functions in each x_i

$$\{\phi_k(x_1)\}_{k=0}^n \otimes \{\psi_j(x_2)\}_{j=0}^n := \{\{\phi_k(x_1)\psi_j(x_2)\}_{j=0}^n\}_{k=0}^n$$

    - Curse of dimensionality: n^m coefficients
- *Complete polynomials*: $\sum_{i=1}^n j_i \leq m$
    - less coefficients, same precision
- *Neural networks*: split function into lower-dimensional ones: $F(x_1, x_2, x_3, x_4) = \tilde{F}(\tilde{f}_1(x_1, x_2), \tilde{f}_2(x_3, x_4))$
- Machine Learning methods: Gaussian Process & RKHS, etc

Sparse grids approach

- If multidimensional function has many cross-partial derivatives

$$\left\| \frac{\partial^{\bar{\alpha}}}{dx_1 \ldots dx_m} f(.) \right\| < \infty$$

for $\bar{\alpha} = \alpha_1 \ldots \alpha_m$ with $\sup_{j \in 1 \ldots m} \alpha_j < k$

- Then interactions are "not too strong"
- Can use many fewer grid points away from axes when interpolating than, e.g., tensor product Chebyshev
- Can efficiently construct *Sparse grids* for interpolation which yield accurate approximation with only polynomial, not exponential dependence on dimension
- Result is *Smolyak* polynomial (or spline) approximation

Reproducing Kernel Hilbert Spaces

- Complete normed space $\mathcal{H}_K$ of functions $F : \mathcal{X} \to \mathbb{R}$ with inner product $\langle , \rangle_K$, in which norm convergence implies pointwise convergence
- Has *reproducing kernel* $k(s,t) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ s.t. $f(s) = \langle f(.), k(s,.)\rangle_K$
- Different k define functions of different shapes, smoothness
  - "Gaussian kernel" $k(s,t) = C\exp(-\frac{d(s,t)^2}{2\sigma^2})$
  - Defined for any metric space $\mathcal{X}$
  - Good for high-dimensional or strange spaces
  - $\mathcal{X}$ can be images, text, other functions, etc
  - Smoothing splines: $\mathcal{H}_K = \mathcal{W}^{2,2k}$ $\langle f, f\rangle_K = \int (f'')^2(x)dx$

Reproducing Kernel Hilbert Spaces: smoothing

- Optimal regression $\inf_{f\in\mathcal{H}_K} \left(\sum_{i=1}^m (y_i - f(x_i))^2 + C\langle f, f\rangle_K\right)$
  - Solution $f = \sum_{i=1}^m a_i k(x_i, .)$ where a_i have simple expression
- Cost is $O(m^3)$, independent of dimension of $\mathcal{X}$
  - Can be improved in some cases
  - If $\mathcal{H}_K$ is smoothing splines, get HP filter
- Equivalent to posterior mean of *Gaussian Process* prior
  - Using data (y_i, x_i) and Gaussian likelihood
- This is prior over functions $f(x)$ where marginals $(f(x_i), f(x_2), \ldots, f(x_n)) \sim N(0, K)$
- $[K]_{ij} := K(x_i, x_j)$ kernel function is covariance matrix of process
- Bayesian interpretation: best guess of function given data
- Also allows calculating uncertainty in approximation

Python tools

- `scipy.interpolate` offers variety of spline methods
  - Linear: `numpy.interp`, `CubicSpline`, B-splines: `make_interp_spline`,
  - Also monotone, 2D, N-D spline on regular, irregular grids
- chebpy or pychebfun: tools for efficient computation with polynomial and basis methods and applications
  - Treat functions as objects, w/ adaptive Chebyshev representations
- Specialized libraries for GPs: GPyTorch, sklearn. gaussian_process, Signal Processing and Wavelets: `scipy.signal`, Smolyak, etc
- To learn, read help and try to replicate with your own code
- You can use these in your research, but not in PS3

Julia tools

- `interpolate()` in `Interpolations.jl`
  - Piecewise constant, linear, quadratic, cubic spline
  - Fast on even grids: use other libraries for alternative splines
  - See QuantEcon for code examples
- ApproxFun system has huge set of tools for efficient computation with polynomial and basis methods and applications
  - Treats functions as objects, w/ adaptive Chebyshev representations
- Wavelets.jl, other specialized packages for GPs, Smolyak, etc
- To learn, read help and try to replicate with your own code
- You can use these in your research, but not in PS3

Matlab tools

- `interp1(), interp2(), interpn()`
  - Piecewise constant, linear, or cubic spline
  - Combines estimation and evaluation steps – very slow
- `spline()` can save estimated coefficients for later use in evaluation
- *CompEcon* has Chebyshev estimation and evaluation tools
- Chebfun system has huge set of tools for efficient computation with Chebyshev methods and applications
- Wavelet toolbox for wavelets, GPStuff for Gaussian Processes
- To learn, read help and try to replicate with your own code
- You can use these in your research, but not in PS3

References 1

- Judd Ch. 6
- Global Polynomials
  - Trefethen, Lloyd N. Approximation Theory and Approximation Practice 2013. `http://www.chebfun.org/ATAP/`
  - Boyd, John P. Chebyshev and Fourier Spectral Methods 2nd ed. Dover, 2000. `http://www-personal.umich.edu/~jpboyd/BOOK_Spectral2000.html`
- Taylor Expansion
  - Fernandez-Villaverde, J., Rubio-Ramirez, J.F., & Schorfheide, F. "Solution and Estimation Methods for DSGE Models" Handbook of Macroeconomics Volume 2A, Ch 9. Elsevier, 2016.

References, ctd

- Spline and RKHS
  - Algorithms and actual code called by Matlab for splines are in:
  - de Boor, Carl. A Practical Guide to Splines Rev. Ed. 2001.
  - Smoothing splines and RKHSs, classic reference:
  - Wahba, Grace. Spline Models for Observational Data 1991.
- Gaussian Process methods:
  - Rasmussen, Carl & Christopher Williams. Gaussian Processes for Machine Learning MIT, 2006. `http://www.gaussianprocess.org/gpml/`
- Wavelets (and Fourier)
  - Mallat, Stephane. A Wavelet Tour of Signal Processing: The Sparse Way, 3rd ed. Elsevier, 2009.
  - G. Peyré. The Numerical Tours of Signal Processing, 2011. `https://www.numerical-tours.com/`