NUMERICAL DYNAMIC PROGRAMMING:

CONTINUOUS STATES

Kenneth L. Judd

Hoover Institution and NBER

July 18, 2006

Continuous Methods for Continuous-State Problems

- Basic Bellman equation:

$$V(x) = \max_{u \in D(x)} \pi(u, x) + \beta\, E\{V(x^+) | x, u)\} \equiv (TV)(x). \tag{12.7.1}$$

  – Discretization essentially approximates V with a step function

  – Approximation theory provides better methods to approximate continuous functions.

- General Task

  – Choose a finite-dimensional parameterization

$$V(x) \doteq \hat{V}(x; a),\ a \in R^m \tag{12.7.2}$$

  and a finite number of states

$$X = \{x_1, x_2, \cdots, x_n\}, \tag{12.7.3}$$

  – Find coefficients $a \in R^m$ such that $\hat{V}(x; a)$ "approximately" satisfies the Bellman equation.

General Parametric Approach: Approximating T

- For each x_j, $(TV)(x_j)$ is defined by

$$v_j = (TV)(x_j) = \max_{u \in D(x_j)} \pi(u, x_j) + \beta \int \hat{V}(x^+; a) dF(x^+ | x_j, u) \tag{12.7.5}$$

- In practice, we compute the approximation $\hat{T}$

$$v_j = (\hat{T}V)(x_j) \doteq (TV)(x_j)$$

  - Integration step: for ω_j and x_j for some numerical quadrature formula

$$\begin{aligned} E\{V(x^+; a) | x_j, u\} &= \int \hat{V}(x^+; a) dF(x^+ | x_j, u) \\ &= \int \hat{V}(g(x_j, u, \varepsilon); a) dF(\varepsilon) \\ &\doteq \sum_{\ell} \omega_\ell \hat{V}(g(x_j, u, \varepsilon_\ell); a) \end{aligned}$$

  - Maximization step: for $x_i \in X$, evaluate

$$v_i = (T\hat{V})(x_i)$$

  - Fitting step:
    * Data: $(v_i, x_i),\ i = 1, \cdots, n$
    * Objective: find an $a \in R^m$ such that $\hat{V}(x; a)$ best fits the data
    * Methods: determined by $\hat{V}(x; a)$

Part I

Approximation Methods

- General Objective: Given data about $f(x)$ construct simpler $g(x)$ approximating $f(x)$.
- Questions:
  - What data should be produced and used?
  - What family of "simpler" functions should be used?
  - What notion of approximation do we use?
- Comparisons with statistical regression
  - Both approximate an unknown function and use a finite amount of data
  - Statistical data is noisy but we assume data errors are small
  - Nature produces data for statistical analysis but we produce the data in function approximation

Interpolation Methods

- Interpolation: find $g(x)$ from an n-D family of functions to exactly fit n data items
- Lagrange polynomial interpolation
  - Data: $(x_i, y_i), i = 1, .., n.$
  - Objective: Find a polynomial of degree $n-1$, $p_n(x)$, which agrees with the data, i.e.,

$$y_i = f(x_i),\ i = 1, .., n$$

  - Result: If the x_i are distinct, there is a unique interpolating polynomial
- Does $p_n(x)$ converge to $f(x)$ as we use more points? Consider $f(x) = \frac{1}{1+x^2}$, x_i uniform on $[-5, 5]$

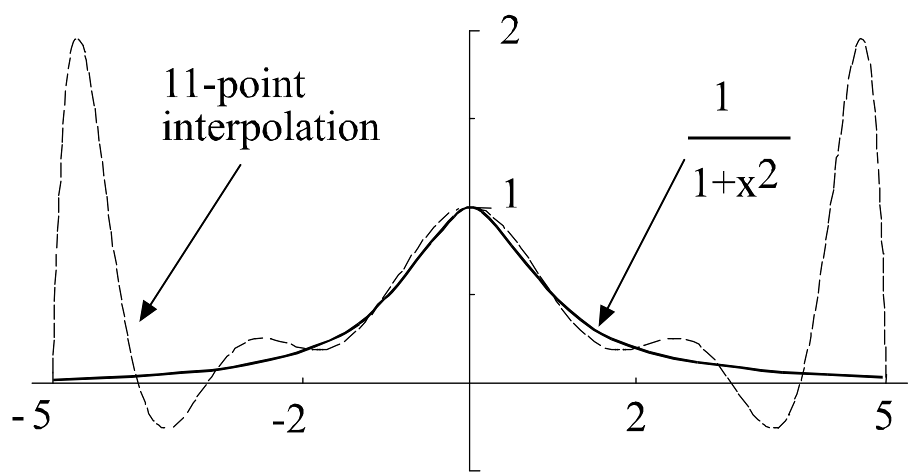


Figure 1:

- Hermite polynomial interpolation
  - Data: $(x_i, y_i, y_i')\,, i = 1, .., n.$
  - Objective: Find a polynomial of degree $2n - 1$, $p(x)$, which agrees with the data, i.e.,

$$
\begin{aligned}
y_i &= p(x_i), \;\; i = 1, .., n \\
y_i' &= p'(x_i), \;\; i = 1, .., n
\end{aligned}
$$

  - Result: If the x_i are distinct, there is a unique interpolating polynomial

- Least squares approximation
  - Data: A function, $f(x)$.
  - Objective: Find a function $g(x)$ from a class G that best approximates $f(x)$, i.e.,

$$
g = \arg\max_{g \in G} \|f - g\|^2
$$

Orthogonal polynomials

- General orthogonal polynomials
  - Space: polynomials over domain D
  - weighting function: $w(x) > 0$
  - Inner product: $\langle f, g \rangle = \int_D f(x)g(x)w(x)dx$
  - Definition: $\{\phi_i\}$ is a family of orthogonal polynomials w.r.t $w(x)$ iff

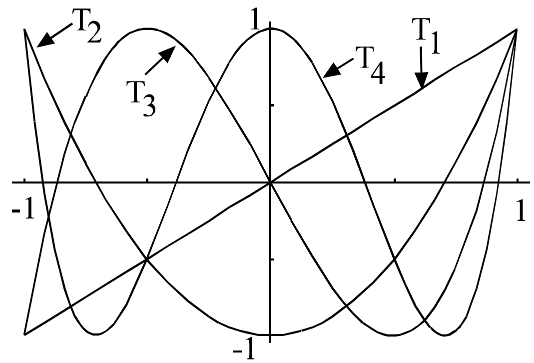
$$\left\langle \phi_i, \phi_j \right\rangle = 0,\ i \neq j$$

  - We like to compute orthogonal polynomials using recurrence formulas

$$\begin{aligned}
\phi_0(x) &= 1 \\
\phi_1(x) &= x \\
\phi_{k+1}(x) &= (a_{k+1}x + b_k)\,\phi_k(x) + c_{k+1}\phi_{k-1}(x)
\end{aligned}$$

- Chebyshev polynomials
  - $[a,b]=[-1,1]$ and $w(x)=\left(1-x^2\right)^{-1/2}$
  - $T_n(x)=\cos(n\cos^{-1}x)$

$$
\begin{aligned}
T_0(x) &= 1\\
T_1(x) &= x\\
T_{n+1}(x) &= 2x\,T_n(x)-T_{n-1}(x),
\end{aligned}
$$



- General Orthogonal Polynomials
  - Few problems have the specific intervals and weights used in definitions
  - One must adapt interval through linear COV: If compact interval $[a,b]$ is mapped to $[-1,1]$ by

$$y=-1+2\frac{x-a}{b-a}$$

then $\phi_i\left(-1+2\frac{x-a}{b-a}\right)$ are orthogonal over $x\in[a,b]$ with respect to $w\left(-1+2\frac{x-a}{b-a}\right)$ iff $\phi_i\left(y\right)$ are orthogonal over $y\in[-1,1]$ w.r.t. $w\left(y\right)$

Regression

- Data: (x_i, y_i), $i = 1, .., n$.
- Objective: Find a function $f(x;\beta)$ with $\beta \in R^m$, $m \leq n$, with $y_i \doteq f(x_i), i = 1, .., n$.
- Least Squares regression:

$$\min_{\beta \in R^m} \sum \left(y_i - f\left(x_i;\beta\right)\right)^2$$

Chebyshev Regression

- Chebyshev Regression Data:
- (x_i, y_i), $i = 1, .., n > m, x_i$ are the n zeroes of $T_n(x)$ adapted to $[a, b]$
- Chebyshev Interpolation Data:

(x_i, y_i), $i = 1, .., n = m, x_i$ are the n zeroes of $T_n(x)$adapted to $[a, b]$

Algorithm 6.4: Chebyshev Approximation Algorithm in $\mathbb{R}^1$

- Objective: Given $f(x)$ defined on $[a, b]$, find its Chebyshev polynomial approximation $p(x)$
- Step 1: Compute the $m \geq n+1$ Chebyshev interpolation nodes on $[-1, 1]$:

$$z_k = -\cos\left(\frac{2k-1}{2m}\ \pi\right)\ ,\ \ k = 1, \cdots, m.$$

- Step 2: Adjust nodes to $[a, b]$ interval:

$$x_k = (z_k + 1)\left(\frac{b-a}{2}\right) + a, k = 1, ..., m.$$

- Step 3: Evaluate f at approximation nodes:

$$w_k = f(x_k)\ ,\ \ k = 1, \cdots, m.$$

- Step 4: Compute Chebyshev coefficients, $a_i, i = 0, \cdots, n$:

$$a_i = \frac{\sum_{k=1}^{m} w_k T_i(z_k)}{\sum_{k=1}^{m} T_i(z_k)^2}$$

to arrive at approximation of $f(x, y)$ on $[a, b]$:

$$p(x) = \sum_{i=0}^{n} a_i T_i\left(2\frac{x-a}{b-a} - 1\right)$$

Minmax Approximation

- Data: $(x_i, y_i), i = 1, .., n.$
- Objective: L^∞ fit

$$\min_{\beta \in R^m} \max_i \|y_i - f(x_i; \beta)\|$$

- Problem: Difficult to compute

- Chebyshev minmax property

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

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

- Chebyshev interpolation:
  - converges in L^∞
  - essentially achieves minmax approximation
  - easy to compute
  - does *not* approximate f'

Splines

Definition 2 *A function $s(x)$ on $[a,b]$ is a spline of order n iff*

1. *s is C^{n-2} on $[a,b]$, and*
2. *there is a grid of points (called nodes) $a = x_0 < x_1 < \cdots < x_m = b$ such that $s(x)$ is a polynomial of degree $n-1$ on each subinterval $[x_i, x_{i+1}]$, $i = 0, \ldots, m-1$.*

Note: an order 2 spline is the piecewise linear interpolant.

- Cubic Splines
  - Lagrange data set: $\{(x_i,\ y_i) \mid i = 0,\ \cdots,\ n\}$.
  - Nodes: The x_i are the nodes of the spline
  - Functional form: $s(x) = a_i + b_i\, x + c_i\, x^2 + d_i\, x^3$ on $[x_{i-1},\ x_i]$
  - Unknowns: $4n$ unknown coefficients, $a_i, b_i, c_i, d_i, i = 1, \cdots n$.

- Conditions:
  - $2n$ interpolation and continuity conditions:

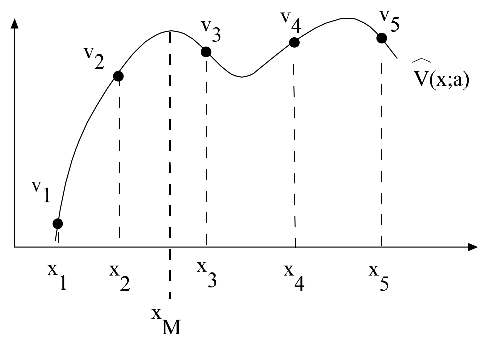
$$
\begin{aligned}
y_i =& a_i + b_i x_i + c_i x_i^2 + d_i x_i^3, \\
& i = 1, ., n \\
y_i =& a_{i+1} + b_{i+1} x_i + c_{i+1} x_i^2 + d_{i+1} x_i^3, \\
& i = 0, ., n-1
\end{aligned}
$$

  - $2n-2$ conditions from C^2 at the interior: for $i = 1, \cdots n-1$,

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

  - Equations (1–4) are $4n-2$ linear equations in $4n$ unknown parameters, a, b, c, and d.
  - construct 2 side conditions:
    - *natural spline:* $s'(x_0) = 0 = s'(x_n)$; it minimizes total curvature, $\int_{x_0}^{x_n} s''(x)^2\, dx$, among solutions to (1-4).
    - *Hermite spline*: $s'(x_0) = y_0'$ and $s'(x_n) = y_n'$ (assumes extra data)
    - *Secant Hermite spline:* $s'(x_0) = (s(x_1) - s(x_0))/(x_1 - x_0)$ and $s'(x_n) = (s(x_n) - s(x_{n-1}))/(x_n - x_{n-1})$.
    - *not-a-knot*: choose $j = i_1, i_2$, such that $i_1 + 1 < i_2$, and set $d_j = d_{j+1}$.
  - Solve system by special (sparse) methods; see spline fit packages

- Shape-preservation
  - Concave (monotone) data may lead to nonconcave (nonmonotone) approximations.
  - Example



- Schumaker Procedure:
  1. Take level (and maybe slope) data at nodes x_i
  2. Add intermediate nodes $z_i^+ \in [x_i, x_{i+1}]$
  3. Run quadratic spline with nodes at the x and z nodes which intepolate data and preserves shape.
  4. Schumaker formulas tell one how to choose the z and spline coefficients (see book and correction at book's website)
- Many other procedures exist for one-dimensional problems, but few procedures exist for two-dimensional problems

- Spline summary:
  - Evaluation is cheap
    * Splines are locally low-order polynomial.
    * Can choose intervals so that finding which $[x_i, x_{i+1}]$ contains a specific x is easy.
    * Finding enclosing interval for general x_i sequence requires at most $\lceil \log_2 n \rceil$ comparisons
  - Good fits even for functions with discontinuous or large higher-order derivatives. E.g., quality of cubic splines depends only on $f^{(4)}(x)$, not $f^{(5)}(x)$.
  - Can use splines to preserve shape conditions

Multidimensional approximation methods

- Lagrange Interpolation
  - Data: $D \equiv \{(x_i, z_i)\}_{i=1}^N \subset R^{n+m}$, where $x_i \in R^n$ and $z_i \in R^m$
  - Objective: find $f : R^n \rightarrow R^m$ such that $z_i = f(x_i)$.
  - Need to choose nodes carefully.
  - Task: Find combinations of interpolation nodes and spanning functions to produce a nonsingular (well-conditioned) interpolation matrix.

Tensor products

- General Approach:
  - If A and B are sets of functions over $x \in R^n$, $y \in R^m$, their tensor product is

$$A \otimes B = \{\varphi(x)\psi(y) \mid \varphi \in A, \psi \in B\}.$$

  - Given a basis for functions of x_i, $\Phi^i = \{\varphi_k^i(x_i)\}_{k=0}^{\infty}$, the *n-fold tensor product* basis for functions of $(x_1, x_2, \ldots, x_n)$ is

$$\Phi = \left\{ \prod_{i=1}^{n} \varphi_{k_i}^i(x_i) \mid k_i = 0, 1, \cdots, \; i = 1, \ldots, n \right\}$$

- Orthogonal polynomials and Least-square approximation
  - Suppose Φ^i are orthogonal with respect to $w_i(x_i)$ over $[a_i, b_i]$
  - Least squares approximation of $f(x_1, \cdots, x_n)$ in Φ is

$$\sum_{\varphi \in \Phi} \frac{\langle \varphi, f \rangle}{\langle \varphi, \varphi \rangle} \varphi,$$

    where the product weighting function

$$W(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} w_i(x_i)$$

    defines $\langle \cdot, \cdot \rangle$ over $D = \prod_i [a_i, b_i]$ in

$$\langle f(x), g(x) \rangle = \int_D f(x)g(x)W(x)dx.$$

Algorithm 6.4: Chebyshev Approximation Algorithm in $\mathbb{R}^2$

- Objective: Given $f(x, y)$ defined on $[a, b] \times [c, d]$, find its Chebyshev polynomial approximation $p(x, y)$
- Step 1: Compute the $m \geq n+1$ Chebyshev interpolation nodes on $[-1, 1]$:

$$z_k = -\cos\left(\frac{2k-1}{2m}\ \pi\right)\ ,\ k = 1, \cdots, m.$$

- Step 2: Adjust nodes to $[a, b]$ and $[c, d]$ intervals:

$$x_k = (z_k + 1)\left(\frac{b-a}{2}\right) + a, k = 1, ..., m.$$

$$y_k = (z_k + 1)\left(\frac{d-c}{2}\right) + c, k = 1, ..., m.$$

- Step 3: Evaluate f at approximation nodes:

$$w_{k,\ell} = f(x_k, y_\ell)\ ,\ k = 1, \cdots, m.\ ,\ \ell = 1, \cdots, m.$$

- Step 4: Compute Chebyshev coefficients, $a_{ij}, i, j = 0, \cdots, n$:

$$a_{ij} = \frac{\sum_{k=1}^{m} \sum_{\ell=1}^{m} w_{k,\ell} T_i(z_k) T_j(z_\ell)}{\left(\sum_{k=1}^{m} T_i(z_k)^2\right)\left(\sum_{\ell=1}^{m} T_j(z_\ell)^2\right)}$$

  to arrive at approximation of $f(x, y)$ on $[a, b] \times [c, d]$:

$$p(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n} a_{ij} T_i\left(2\frac{x-a}{b-a} - 1\right) T_j\left(2\frac{y-c}{d-c} - 1\right)$$

Multidimensional Splines

- B-splines: Multidimensional versions of splines can be constructed through tensor products; here B-splines would be useful.
- Summary
  - Tensor products directly extend one-dimensional methods to n dimensions
  - Curse of dimensionality often makes tensor products impractical

Complete polynomials

- Taylor's theorem for $\mathbb{R}^n$ produces the approximation

$$f(x) \doteq f(x^0) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(x^0)\ (x_i - x_i^0)$$

$$+\frac{1}{2}\sum_{i_1=1}^{n}\sum_{i_2=1}^{n}\frac{\partial^2 f}{\partial x_{i_1}\partial x_{i_k}}(x_0)(x_{i_1} - x_{i_1}^0)(x_{i_k} - x_{i_k}^0) + ...$$

  - For $k = 1$, Taylor's theorem for n dimensions used the linear functions $\mathcal{P}_1^n \equiv \{1, x_1, x_2, \cdots, x_n\}$
  - For $k = 2$, Taylor's theorem uses $\mathcal{P}_2^n \equiv \mathcal{P}_1^n \cup \{x_1^2, \cdots, x_n^2, x_1x_2, x_1x_3, \cdots, x_{n-1}x_n\}$.
- In general, the kth degree expansion uses the *complete set of polynomials of total degree k in n variables.*

$$\mathcal{P}_k^n \equiv \{x_1^{i_1} \cdots x_n^{i_n} \mid \sum_{\ell=1}^{n} i_\ell \leq k,\ 0 \leq i_1, \cdots, i_n\}$$

- Complete orthogonal basis includes only terms with total degree k or less.
- Sizes of alternative bases

degree k	$\mathcal{P}_k^n$	Tensor Prod.
2	$1+n+n(n+1)/2$	3^n
3	$1+n+\frac{n(n+1)}{2}+n^2+\frac{n(n-1)(n-2)}{6}$	4^n

  - Complete polynomial bases contains fewer elements than tensor products.
  - Asymptotically, complete polynomial bases are as good as tensor products.
  - For smooth n-dimensional functions, complete polynomials are more efficient approximations

- Construction
  - Compute tensor product approximation, as in Algorithm 6.4
  - Drop terms not in complete polynomial basis (or, just compute coefficients for polynomials in complete basis).
  - Complete polynomial version is faster to compute since it involves fewer terms

Aproximation Summary

- Need to find flexible but efficient way to approximate unknown functions
- Classical methods
  - Polynomials, orthogonal polynomials
  - Splines
- New methods
  - Neural nets
  - Radial basis functions
- Be "imaginative"
  - Nonlinear change of variables
  - Build your own basis reflecting knowledge of solution.

Part II

Integration

- Most integrals cannot be evaluated analytically
- Integrals frequently arise in economics
  - Expected utility and discounted utility and profits over a long horizon
  - Bayesian posterior
  - Solution methods for dynamic economic models

Gaussian Formulas

- All integration formulas choose *quadrature nodes* $x_i \in [a, b]$ and *quadrature weights* ω_i:

$$\int_a^b f(x)\,dx \doteq \sum_{i=1}^{n} \omega_i f(x_i) \tag{7.2.1}$$

  - Newton-Cotes (trapezoid, Simpson, etc.) use arbitrary x_i
  - Gaussian quadrature uses good choices of x_i nodes and ω_i weights.
- Exact quadrature formulas:
  - Let $\mathcal{F}_k$ be the space of degree k polynomials
  - A quadrature formula is exact of degree k if it correctly integrates each function in $\mathcal{F}_k$
  - Gaussian quadrature formulas use n points and are exact of degree $2n - 1$

Theorem 3 *Suppose that* $\{\varphi_k(x)\}_{k=0}^{\infty}$ *is an orthonormal family of polynomials with respect to* $w(x)$ *on* $[a,\, b]$. *Then there are* x_i *nodes and weights* ω_i *such that* $a < x_1 < x_2 < \cdots < x_n < b$, *and*

1. *if* $f \in C^{(2n)}[a,\, b]$, *then for some* $\xi \in [a,\, b]$,

$$\int_a^b w(x)\, f(x)\; dx = \sum_{i=1}^{n} \omega_i\, f(x_i) + \frac{f^{(2n)}(\xi)}{q_n^2 (2n)!};$$

2. *and* $\sum_{i=1}^{n} \omega_i f(x_i)$ *is the unique formula on* n *nodes that exactly integrates* $\int_a^b f(x)\, w(x)\, dx$ *for all polynomials in* $\mathcal{F}_{2n-1}$.

Gauss-Chebyshev Quadrature

- Domain: $[-1, 1]$
- Weight: $(1 - x^2)^{-1/2}$
- Formula:

$$\int_{-1}^{1} f(x)(1 - x^2)^{-1/2}\, dx = \frac{\pi}{n} \sum_{i=1}^{n} f(x_i) + \frac{\pi}{2^{2n-1}} \frac{f^{(2n)}(\xi)}{(2n)!} \tag{7.2.4}$$

for some $\xi \in [-1, 1]$, with quadrature nodes

$$x_i = cos\left(\frac{2i - 1}{2n}\pi\right), \quad i = 1, ..., n. \tag{7.2.5}$$

Arbitrary Domains

- Want to approximate $\int_a^b f(x)\, dx$ for different range, and/or no weight function
  - Linear change of variables $x = -1 + 2(y - a)(b - a)$
  - Multiply the integrand by $(1 - x^2)^{1/2} / (1 - x^2)^{1/2}$.

$$\int_a^b f(y)\ dy = \frac{b - a}{2} \int_{-1}^{1} f\left(\frac{(x + 1)(b - a)}{2} + a\right) \frac{\left(1 - x^2\right)^{1/2}}{(1 - x^2)^{1/2}}\ dx$$

  - Gauss-Chebyshev quadrature uses the x_i Gauss-Chebyshev nodes over $[-1, 1]$

$$\int_a^b f(y)\ dy \doteq \frac{\pi(b - a)}{2n} \sum_{i=1}^{n} f\left(\frac{(x_i + 1)(b - a)}{2} + a\right) \left(1 - x_i^2\right)^{1/2}$$

Gauss-Hermite Quadrature

- Domain is $[-\infty, \infty]$ and weight is e^{-x^2}
- Formula: for some $\xi \in (-\infty, \infty)$.

$$\int_{-\infty}^{\infty} f(x)e^{-x^2} dx = \sum_{i=1}^{n} \omega_i f(x_i) + \frac{n!\sqrt{\pi}}{2^n} \cdot \frac{f^{(2n)}(\xi)}{(2n)!}$$

N	x_i	ω_i	N	x_i	ω_i
2	0.7071067811	0.8862269254	7	0.2651961356(1)	0.9717812450(−3)
				0.1673551628(1)	0.5451558281(−1)
3	0.1224744871(1)	0.2954089751		0.8162878828	0.4256072526
	0.0000000000	0.1181635900(1)		0.0000000000	0.8102646175

- Normal Random Variables
  - Y is distributed $N(\mu, \sigma^2)$. Expectation is integration.
  - Use Gauss-Hermite quadrature: Linear COV $x = (y - \mu)/\sqrt{2}\,\sigma$ implies

$$E\{f(Y)\} = \int_{-\infty}^{\infty} f(y)e^{-(y-\mu)^2/(2\sigma^2)}\, dy = \int_{-\infty}^{\infty} f(\sqrt{2}\,\sigma\, x + \mu)e^{-x^2}\sqrt{2}\,\sigma\, dx$$

$$\doteq \pi^{-\frac{1}{2}} \sum_{i=1}^{n} \omega_i f(\sqrt{2}\,\sigma\, x_i + \mu)$$

where the ω_i and x_i are the Gauss-Hermite quadrature weights and nodes over $[-\infty, \infty]$.

Multidimensional Integration

- Most economic problems have several dimensions
  - Multiple assets
  - Multiple error terms
- Multidimensional integrals are much more difficult
  - Simple methods suffer from curse of dimensionality
  - There are methods which avoid curse of dimensionality

Product Rules

- Build product rules from one-dimension rules
- Let x_i^ℓ, ω_i^ℓ, $\quad i = 1, \cdots, m$, be one-dimensional quadrature points and weights in dimension ℓ from a Newton-Cotes rule or the Gauss-Legendre rule.
- The *product rule*

$$\int_{[-1,1]^d} f(x)dx \doteq \sum_{i_1=1}^{m} \cdots \sum_{i_d=1}^{m} \omega_{i_1}^1 \omega_{i_2}^2 \cdots \omega_{i_d}^d \, f(x_{i_1}^1, x_{i_2}^2, \cdots, x_{i_d}^d)$$

- Gaussian structure prevails
  - Suppose $w^\ell(x)$ is weighting function in dimension ℓ
  - Define the d-dimensional weighting function.

$$W(x) \equiv W(x_1, \cdots, x_d) = \prod_{\ell=1}^{d} w^\ell(x_\ell)$$

  - Product Gaussian rules are based on product orthogonal polynomials.
- Curse of dimensionality:
  - m^d functional evaluations is m^d for a d-dimensional problem with m points in each direction.
  - Problem worse for Newton-Cotes rules which are less accurate in $\mathbb{R}^1$.

Monomial Formulas: A Nonproduct Approach

- Method
- Choose $x^i \in D \subset \mathbb{R}^d,\ i = 1, ..., N$
- Choose $\omega_i \in \mathbb{R},\ i = 1, ..., N$
- Quadrature formula

$$\int_D f(x)\, dx \doteq \sum_{i=1}^{N} \omega_i\, f(x^i) \tag{7.5.3}$$

- A monomial formula is complete for degree ℓ if

$$\sum_{i=1}^{N} \omega_i\, p(x^i) = \int_D p(x)\, dx \tag{7.5.3}$$

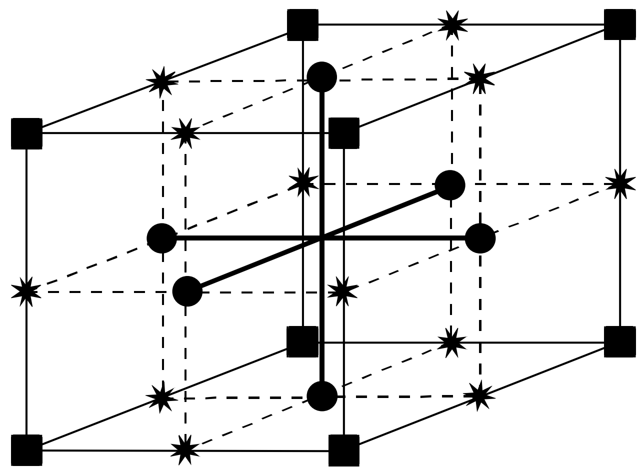
  for all polynomials $p(x)$ of total degree ℓ; recall that $\mathcal{P}_\ell$ was defined in chapter 6 to be the set of such polynomials.

- For the case $\ell = 2$, this implies the equations

$$\begin{array}{ll} \sum_{i=1}^{N} \omega_i & = \int_D 1 \cdot dx \\ \sum_{i=1}^{N} \omega_i x_j^i & = \int_D x_j\, dx,\ j = 1, \cdots, d \\ \sum_{i=1}^{N} \omega_i x_j^i x_k^i & = \int_D x_j x_k\, dx,\ j, k = 1, \cdots, d \end{array} \tag{7.5.4}$$

  - $1 + d + \frac{1}{2} d(d+1)$ equations
  - N weights ω_i and the N nodes x^i each with d components, yielding a total of $(d+1)N$ unknowns.

Quadrature Node Sets



- Natural types of nodes:
  - The center
  - The circles: centers of faces
  - The stars: centers of edges
  - The squares: vertices
- Some monomial formulas will take some combinations of these sets
- Other types of collections are possible

- Simple examples
  - Let $e^j \equiv (0, \ldots, 1, \ldots, 0)$ where the '1' appears in column j.
  - $2d$ points and exactly integrates all elements of $\mathcal{P}_3$ over $[-1,1]^d$

$$\int_{[-1,1]^d} f \doteq \omega \sum_{i=1}^{d} \left(f(ue^i) + f(-ue^i) \right)$$

$$u = \left(\frac{d}{3}\right)^{1/2}, \quad \omega = \frac{2^{d-1}}{d}$$

  - For $\mathcal{P}_5$ the following scheme works:

$$\begin{aligned}\int_{[-1,1]^d} f \doteq & \omega_1 f(0) + \omega_2 \sum_{i=1}^{d} \left(f(ue^i) + f(-ue^i) \right) \\ & + \omega_3 \sum_{\substack{1 \le i < d, \\ i < j \le d}} \left(f(u(e^i \pm e^j)) + f(-u(e^i \pm e^j)) \right)\end{aligned}$$

where

$$\begin{aligned}\omega_1 &= 2^d(25\ d^2 - 115\ d + 162), \quad \omega_2 = 2^d(70 - 25d) \\ \omega_3 &= \frac{25}{324}\ 2^d, \quad u = \left(\frac{3}{5}\right)^{1/2}.\end{aligned}$$

Integration Summary

- Classical methods
  - Trapezoid, Simpson rules
  - Gaussian quadrature rules
  - Product rules for m
- Multidimensional integrals
  - Product rules of one-dimensional rules could be used
  - Product rules create curse of dimensionality
  - Monte Carlo integration is too imprecise
  - Monomial rules are much more efficient
  - There is a large literature on quadrature that can be applied to economics problems
  - Computers can construct integration rules that are best for your problem.
  - *There is no curse of dimensionality!*

Part III

General Parametric Approach for Dynamic Programming

- For each x_j, $(TV)(x_j)$ is defined by

$$v_j = (TV)(x_j) = \max_{u \in D(x_j)} \pi(u, x_j) + \beta \int \hat{V}(x^+; a) dF(x^+|x_j, u) \tag{12.7.5}$$

- In practice, we compute the approximation $\hat{T}$

$$v_j = (\hat{T}V)(x_j) \doteq (TV)(x_j)$$

  – Integration step: for ω_j and x_j for some numerical quadrature formula

$$\begin{aligned} E\{V(x^+; a)|x_j, u)\} &= \int \hat{V}(x^+; a) dF(x^+|x_j, u) \\ &= \int \hat{V}(g(x_j, u, \varepsilon); a) dF(\varepsilon) \\ &\doteq \sum_{\ell} \omega_{\ell} \hat{V}(g(x_j, u, \varepsilon_{\ell}); a) \end{aligned}$$

  – Maximization step: for $x_i \in X$, evaluate

$$v_i = (T\hat{V})(x_i)$$

  – Fitting step:
    * Data: $(v_i, x_i),\ i = 1, \cdots, n$
    * Objective: find an $a \in R^m$ such that $\hat{V}(x; a)$ best fits the data
    * Methods: determined by $\hat{V}(x; a)$

Approximating T with Hermite Data

- Conventional methods just generate data on $V(x_j)$:

$$v_j = \max_{u \in D(x_j)} \pi(u, x_j) + \beta \int \hat{V}(x^+; a) dF(x^+ | x_j, u) \tag{12.7.5}$$

- Envelope theorem:
  - If solution u is interior,

$$v_j' = \pi_x(u, x_j) + \beta \int \hat{V}(x^+; a) dF_x(x^+ | x_j, u)$$

  - If solution u is on boundary

$$v_j' = \mu + \pi_x(u, x_j) + \beta \int \hat{V}(x^+; a) dF_x(x^+ | x_j, u)$$

    where μ is a Kuhn-Tucker multiplier

- Since computing v_j' is cheap, we should include it in data:
  - Data: $(v_i, v_i', x_i),\ i = 1, \cdots, n$
  - Objective: find an $a \in R^m$ such that $\hat{V}(x; a)$ best fits Hermite data
  - Methods: determined by $\hat{V}(x; a)$

General Parametric Approach: Value Function Iteration

$$\text{guess } a \longrightarrow \hat{V}(x;a) \longrightarrow (v_i, x_i),\ i=1,\cdots,n \longrightarrow \text{new } a$$

- Comparison with discretization
  - This procedure examines only a finite number of points, but does *not* assume that future points lie in same finite set.
  - Our choices for the x_i are guided by systematic numerical considerations.
- Synergies
  - Smooth interpolation methods allow us to use efficient quadrature rules in the integral in (12.7.5)
  - Smooth interpolation methods Newton's method in the maximization step.
  - They also make it easier to evaluate the integral in (12.7.5).
- Finite-horizon problems
  - Value function iteration is only possible procedure since $V(x,t)$ depends on time t.
  - Begin with terminal value function, $V(x,T)$
  - Compute approximations for each $V(x,t)$, $t = T-1, T-2$, etc.

Algorithm 12.5: Parametric Dynamic Programming
with Value Function Iteration

Objective: Solve the Bellman equation, (12.7.1).

Step 0: Choose functional form $\hat{V}(x;a)$ and grid, $X = \{x_1, ..., x_n\}$.
Make initial guess $\hat{V}(x;a^0)$, and choose stopping criterion $\epsilon > 0$.

Step 1: Maximization step: Compute $v_j = (T\hat{V}(\cdot;a^i))(x_j)$ for all $x_j \in X$.

Step 2: Fitting step: Using the appropriate approximation method, compute the $a^{i+1} \in R^m$ such that $\hat{V}(x;a^{i+1})$ approximates the (v_i, x_i) data.

Step 3: If $\| \hat{V}(x;a^i) - \hat{V}(x;a^{i+1}) \| < \epsilon$, STOP; else go to step 1.

- Convergence
  - T is a contraction mapping
  - $\hat{T}$ may be neither monotonic nor a contraction
- Shape problems
  - An instructive example

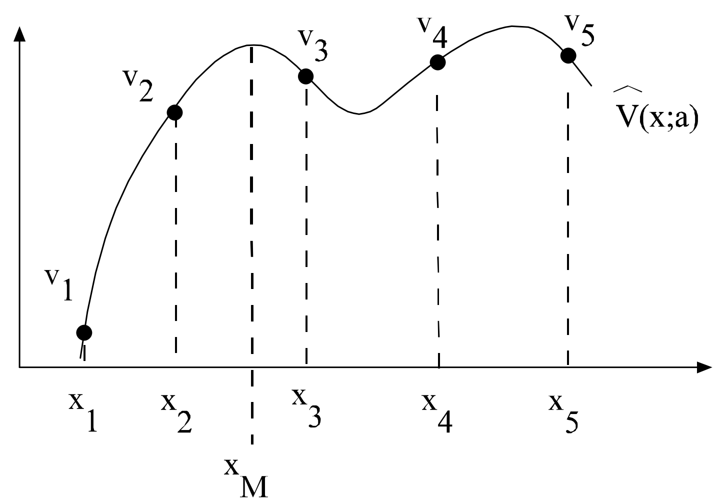


Figure 2:

  - Shape problems may become worse with value function iteration
- The true approximation problem:
  - Take data and find "nearest" function that replicates shape of data
  - In concave case, this is regression on to the cone of concave functions

Nonlinear Programming Approach

- Reformulate dynamic programming as a nonlinear programming problem (Judd and Su)
  - Infinitistic version

$$\begin{array}{l} \max \int V\left(x\right)dx \\ s.t.\ \ V\left(x\right) \leq \pi(x,u) + \beta E\left\{V\left(x\right)|x,u\right\},\ \forall x,u, \end{array}$$

  - Finite-dimensional approximation:
    - Approximate value function

$$V\left(x;a\right) = \sum_{i=1}^{n} a_i \phi_i\left(x\right)$$

    - Choose coefficients

$$\begin{array}{ll} \max_{u_j,a} \sum_{j=1}^{m} V\left(x_j;a\right) & \\ s.t. & V\left(x_j\right) \leq \pi(x_j,u_j) + \beta E\left\{V\left(x^+\right)|x_j,u_j\right\},\ \forall j, \end{array}$$

- Experience:
  - Frequent instabilities when using ordinary polynomials
  - Much better performance if we impose shape restrictions

$$V'\left(x_j\right) > 0$$
$$V''\left(x_j\right) < 0$$

Summary:

- Discretization methods
  - Easy to implement
  - Numerically stable
  - Amenable to many accelerations
  - Poor approximation to continuous problems
- Continuous approximation methods
  - Can exploit smoothness in problems
  - Possible numerical instabilities
  - Acceleration is less possible