Numerical Dynamic Programming

Kenneth L. Judd
Hoover Institution

Prepared for ICE05
July 20, 2005 - lectures

Discrete-Time Dynamic Programming

- Objective:

$$E\left\{\sum_{t=1}^{T}\pi(x_t,\,u_t,\,t)+W(x_{T+1})\right\}, \tag{12.1.1}$$

  - X: set of states
  - $\mathcal{D}$: the set of controls
  - $\pi(x,\,u,\,t)$ payoffs in period t, for $x\in X$ at the beginning of period t, and control $u\in\mathcal{D}$ is applied in period t.
  - $D(x,t)\subseteq\mathcal{D}$: controls which are feasible in state x at time t.
  - $F(A;x,u,t)$: probability that $x_{t+1}\in A\subset X$ conditional on time t control and state

- Value function

$$V(x,t)\equiv\sup_{\mathcal{U}(x,t)}E\left\{\sum_{s=t}^{T}\pi(x_s,\,u_s,\,s)+W(x_{T+1})|x_t=x\right\}. \tag{12.1.2}$$

- Bellman equation

$$V(x,t)=\sup_{u\in D(x,t)}\ \pi(x,\,u,\,t)+E\left\{V(x_{t+1},\,t+1)|x_t=x,u_t=u\right\} \tag{12.1.3}$$

- Existence: boundedness of π is sufficient

Autonomous, Infinite-Horizon Problem:

- Objective:

$$\max_{u_t} E\left\{\sum_{t=1}^{\infty} \beta^t \pi(x_t,\, u_t)\right\} \tag{12.1.1}$$

  - X: set of states
  - $\mathcal{D}$: the set of controls
  - $D(x) \subseteq \mathcal{D}$: controls which are feasible in state x.
  - $\pi(x,\, u)$ payoff in period t if $x \in X$ at the beginning of period t, and control $u \in \mathcal{D}$ is applied in period t.
  - $F(A; x, u)$: probability that $x^+ \in A \subset X$ conditional on current control u and current state x.

- Value function definition: if $\mathcal{U}(x)$ is set of all feasible strategies starting at x.

$$V(x) \equiv \sup_{\mathcal{U}(x)} E\left\{\sum_{t=0}^{\infty} \beta^t \pi(x_t,\, u_t) \middle| x_0 = x\right\}, \tag{12.1.8}$$

- Bellman equation for $V(x)$

$$V(x) = \sup_{u \in D(x)} \pi(x,\, u) + \beta\, E\, \left\{ V(x^+) | x, u \right\} \equiv (TV)(x), \qquad (12.1.9)$$

- Optimal policy function, $U(x)$, if it exists, is defined by

$$U(x) \;\in\; \arg \max_{u \in D(x)} \pi(x,\, u) + \beta\, E\, \left\{ V(x^+) | x, u \right\}$$

- Standard existence theorem:

Theorem 1 *If X is compact, $\beta < 1$, and π is bounded above and below, then the map*

$$TV = \sup_{u \in D(x)} \pi(x, u) + \beta E \left\{ V(x^+) \mid x, u \right\} \qquad (12.1.10)$$

is monotone in V, is a contraction mapping with modulus β in the space of bounded functions, and has a unique fixed point.

Deterministic Growth Example

- Problem:

$$\begin{aligned} V(k_0) = \max_{c_t} & \sum_{t=0}^{\infty} \beta^t u(c_t), \\ & k_{t+1} = F(k_t) - c_t \\ & k_0 \text{ given} \end{aligned} \tag{12.1.12}$$

  - Euler equation:

$$u'(c_t) = \beta u'(c_{t+1}) F'(k_{t+1})$$

  - Bellman equation

$$V(k) = \max_c \; u(c) + \beta V(F(k) - c). \tag{12.1.13}$$

  - Solution to (12.1.12) is a policy function $C(k)$ and a value function $V(k)$ satisfying

$$0 = u'(C(k))F'(k) - V'(k) \tag{12.1.15}$$

$$V(k) = u(C(k)) + \beta V(F(k) - C(k)) \tag{12.1.16}$$

- (12.1.16) defines the value of an arbitrary policy function $C(k)$, not just for the optimal $C(k)$.
- The pair (12.1.15) and (12.1.16)
  - expresses the value function given a policy, and
  - a first-order condition for optimality.

Stochastic Growth Accumulation

- Problem:

$$V(k,\theta) = \max_{c_t,\ell_t} \; E\left\{\sum_{t=0}^{\infty} \beta^t \; u(c_t)\right\}$$
$$k_{t+1} = F(k_t,\theta_t) - c_t$$
$$\theta_{t+1} = g(\theta_t,\varepsilon_t)$$
$$\varepsilon_t : \text{ i.i.d. random variable}$$
$$k_0 = k, \;\; \theta_0 = \theta.$$

- State variables:
  - k: productive capital stock, endogenous
  - θ: productivity state, exogenous

- The dynamic programming formulation is

$$V(k,\theta) = \max_{c} \; u(c) + \beta E\{V(F(k,\theta) - c, \theta^+)|\theta\} \qquad (12.1.21)$$
$$\theta^+ = g(\theta,\varepsilon)$$

- The control law $c = C(k,\theta)$ satisfies the first-order conditions

$$0 = u_c\left(C(k,\theta)\right) - \beta \, E\left\{u_c(C(k^+,\theta^+))F_k(k^+,\theta^+) \mid \theta\right\}, \qquad (12.1.23)$$

where

$$k^+ \equiv F(k, L(k,\theta),\theta) - C(k,\theta),$$

Objectives of this Lecture

- Formulate dynamic programming problems in computationally useful ways
- Describe key algorithms
  - Value function iteration
  - Policy iteration
  - Gauss-Seidel methods
  - Linear programming approach
- Describe approaches to continuous-state problems
  - Point to key numerical steps
    - Approximation of value functions
    - Numerical integration methods
    - Maximization methods
  - Describe how to combine alternative numerical techniques and algorithms

Discrete State Space Problems

- State space $X = \{x_i,\ i = 1, \cdots, n\}$
- Controls $\mathcal{D} = \{u_i | i = 1, ..., m\}$
- $q_{ij}^t(u) = \Pr\left(x_{t+1} = x_j | x_t = x_i, u_t = u\right)$
- $Q^t(u) = \left(q_{ij}^t(u)\right)_{i,j}$: Markov transition matrix at t if $u_t = u$.

Value Function iteration

- Terminal value:

$$V_i^{T+1} = W(x_i),\ i = 1, \cdots, n.$$

- Bellman equation: time t value function is

$$V_i^t = \max_u \left[\pi(x_i, u, t) + \beta \sum_{j=1}^n q_{ij}^t(u)\, V_j^{t+1}\right],\ i = 1, \cdots, n$$

  which defines *value function iteration*

- Value function iteration is only choice for finite-horizon problems
- Infinite-horizon problems
  - Bellman equation is a set of equations for V_i values:

$$V_i = \max_u \left[\pi(x_i, u) + \beta \sum_{j=1}^n q_{ij}(u)\, V_j\right],\ i = 1, \cdots, n$$

  - Value function iteration is now

$$V_i^{k+1} = \max_u \left[\pi(x_i, u) + \beta \sum_{j=1}^n q_{ij}(u)\, V_j^k\right],\ i = 1, \cdots, n$$

– Can use value function iteration with arbitrary V_i^0 and iterate $k \to \infty$.

– Error is given by contraction mapping property:

$$\left\|V^k - V^*\right\| \leq \frac{1}{1-\beta}\left\|V^{k+1} - V^k\right\|$$

Algorithm 12.1: Value Function Iteration Algorithm

Objective: Solve the Bellman equation, (12.3.4).

Step 0: Make initial guess V^0; choose stopping criterion $\epsilon > 0$.

Step 1: For $i = 1, ..., n$, compute
$V_i^{\ell+1} = \max_{u \in D} \ \pi(x_i, u) + \beta \sum_{j=1}^n q_{ij}(u) V_j^\ell$.

Step 2: If $\| V^{\ell+1} - V^\ell \| < \epsilon$, then go to step 3; else go to step 1.

Step 3: Compute the final solution, setting
$U^* = \mathcal{U} V^{\ell+1}$,
$P_i^* = \pi(x_i, U_i^*), \qquad i = 1, \cdots, n,$
$V^* = (I - \beta Q^{U^*})^{-1} P^*$,
and STOP.

Output:

Policy Iteration (a.k.a. Howard improvement)

- Value function iteration is a slow process
  - Linear convergence at rate β
  - Convergence is particularly slow if β is close to 1.
- Policy iteration is faster
  - Current guess:
$$V_i^k,\ i = 1, \cdots, n.$$
  - Iteration: compute optimal policy today if V^k is value tomorrow:
$$U_i^{k+1} = \arg\max_u \left[\pi(x_i, u) + \beta \sum_{j=1}^{n} q_{ij}(u)\, V_j^k \right],\ i = 1, \cdots, n,$$
  - Compute the value function if the policy U^{k+1} is used forever, which is solution to the linear system
$$V_i^{k+1} = \pi\left(x_i, U_i^{k+1}\right) + \beta \sum_{j=1}^{n} q_{ij}(U_i^{k+1})\, V_j^{k+1},\ i = 1, \cdots, n,$$
- Comments:
  - Policy iteration depends on only monotonicity
  - Policy iteration is faster than value function iteration
    - If initial guess is above or below solution then policy iteration is between truth and value function iterate
    - Works well even for β close to 1.

Gaussian acceleration methods for infinite-horizon models

- Key observation: Bellman equation is a simultaneous set of equations

$$V_i = \max_u \left[\pi(x_i, u) + \beta \sum_{j=1}^{n} q_{ij}(u)\, V_j \right], \; i = 1, \cdots, n$$

- Idea: Treat problem as a large system of nonlinear equations
- Value function iteration is the *pre-Gauss-Jacobi* iteration

$$V_i^{k+1} = \max_u \left[\pi(x_i, u) + \beta \sum_{j=1}^{n} q_{ij}(u)\, V_j^k \right], \; i = 1, \cdots, n$$

- True Gauss-Jacobi is

$$V_i^{k+1} = \max_u \left[\frac{\pi(x_i, u) + \beta \sum_{j \neq i} q_{ij}(u)\, V_j^k}{1 - \beta q_{ii}(u)} \right], \; i = 1, \cdots, n$$

- pre-Gauss-Seidel iteration
  - Value function iteration is a pre-Gauss-Jacobi scheme.
  - Gauss-Seidel alternatives use new information immediately
    - Suppose we have V_i^ℓ
    - At each x_i, given $V_j^{\ell+1}$ for $j < i$, compute $V_i^{\ell+1}$ in a pre-Gauss-Seidel fashion

$$V_i^{\ell+1} = \max_u \; \pi(x_i, u) + \beta \sum_{j<i} q_{ij}(u) V_j^{\ell+1} + \beta \sum_{j \geq i} q_{ij}(u) V_j^\ell \quad (12.4.7)$$

    - Iterate (12.4.7) for $i = 1, .., n$

- Gauss-Seidel iteration
  - Suppose we have V_i^ℓ
  - If optimal control at state i is u, then Gauss-Seidel iterate would be

$$V_i^{\ell+1} = \pi(x_i, u) + \beta \frac{\sum_{j<i} q_{ij}(u) V_j^{\ell+1} + \sum_{j>i} q_{ij}(u) V_j^\ell}{1 - \beta q_{ii}(u)}$$

  - Gauss-Seidel: At each x_i, given $V_j^{\ell+1}$ for $j < i$, compute $V_i^{\ell+1}$

$$V_i^{\ell+1} = \max_u \frac{\pi(x_i, u) + \beta \sum_{j<i} q_{ij}(u) V_j^{\ell+1} + \beta \sum_{j>i} q_{ij}(u) V_j^\ell}{1 - \beta q_{ii}(u)}$$

  - Iterate this for $i = 1, .., n$

- Gauss-Seidel iteration: better notation
  - No reason to keep track of ℓ, number of iterations
  - At each x_i,

$$V_i \longleftarrow \max_u \frac{\pi(x_i, u) + \beta \sum_{j<i} q_{ij}(u) V_j + \beta \sum_{j>i} q_{ij}(u) V_j}{1 - \beta q_{ij}(u)}$$

  - Iterate this for $i = 1, .., n, 1, ..., $ etc.

Upwind Gauss-Seidel

- Gauss-Seidel methods in (12.4.7) and (12.4.8)
  - Sensitive to ordering of the states.
  - Need to find good ordering schemes to enhance convergence.
- Example:
  - Two states, x_1 and x_2, and two controls, u_1 and u_2
    - u_i causes state to move to x_i, $i = 1, 2$
    - Payoffs:

$$\begin{aligned} \pi(x_1, u_1) &= -1, \ \pi(x_1, u_2) = 0, \\ \pi(x_2, u_1) &= 0, \ \pi(x_2, u_2) = 1. \end{aligned} \tag{12.4.9}$$

    - $\beta = 0.9$.
  - Solution:
    - Optimal policy: always choose u_2, moving to x_2
    - Value function:

$$V(x_1) = 9, \ V(x_2) = 10.$$

    - x_2 is the unique steady state, and is stable
  - Value iteration with $V^0(x_1) = V^0(x_2) = 0$ converges linearly:

$$\begin{aligned} &V^1(x_1) = 0, \ V^1(x_2) = 1, \ U^1(x_1) = 2, \ U^1(x_2) = 2, \\ &V^2(x_1) = 0.9, \ V^2(x_2) = 1.9, \ U^2(x_1) = 2, \ U^2(x_2) = 2, \\ &V^3(x_1) = 1.71, \ V^3(x_2) = 2.71, \ U^3(x_1) = 2, \ U^3(x_2) = 2, \end{aligned}$$

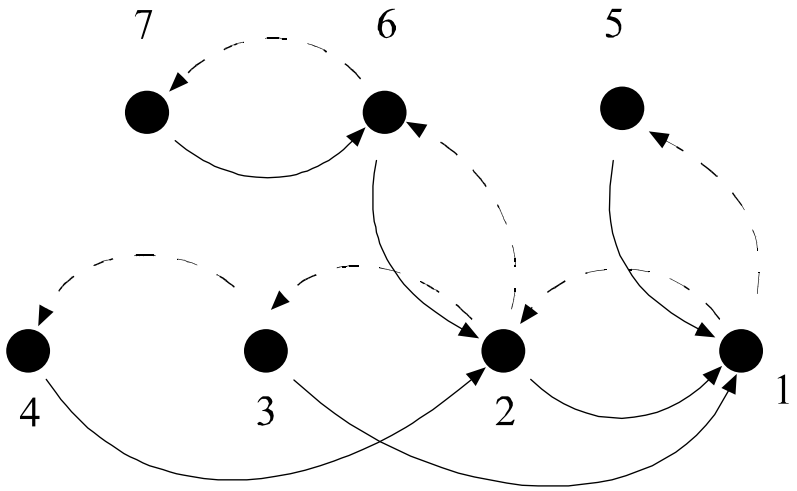
  - Policy iteration converges after two iterations

$$\begin{aligned} &V^1(x_1) = 0, \ V^1(x_2) = 1, \ U^1(x_1) = 2, \ U^1(x_2) = 2, \\ &V^2(x_1) = 9, \ V^2(x_2) = 10, \ U^2(x_1) = 2, \ U^2(x_2) = 2, \end{aligned}$$

- Upwind Gauss-Seidel
  - Value function at absorbing states is trivial to compute
    - Suppose s is absorbing state with control u
    - $V(s) = \pi(s,u)/(1-\beta)$.
  - With absorbing state $V(s)$ we compute $V(s')$ of any s' that sends system to s.

$$V(s') = \pi(s', u) + \beta V(s)$$

  - With $V(s')$, we can compute values of states s'' that send system to s'; etc.



- Alternating Sweep
  - It may be difficult to find proper order.
  - Idea: alternate between two approaches with different directions.

$$\begin{aligned} W &= V^k, \\ W_i &= \max_u \ \pi(x_i, u) + \beta \textstyle\sum_{j=1}^n q_{ij}(u) W_j, \ i = 1, 2, 3, ..., n \\ W_i &= \max_u \ \pi(x_i, u) + \beta \textstyle\sum_{j=1}^n q_{ij}(u) W_j, \ i = n, n-1, ..., 1 \\ V^{k+1} &= W \end{aligned}$$

  - Will always work well in one-dimensional problems since state moves either right or left, and alternating sweep will exploit this half of the time.
  - In two dimensions, there may still be a natural ordering to be exploited.

- Simulated Upwind Gauss-Seidel
  - It may be difficult to find proper order in higher dimensions
  - Idea: simulate using latest policy function to find downwind direction
    * Simulate to get an example path, $x_1, x_2, x_3, x_4, ..., x_m$
    * Execute Gauss-Seidel with states $x_m, x_{m-1}, x_{m-2}, ..., x_1$

Linear Programming Approach

- If $\mathcal{D}$ is finite, we can reformulate dynamic programming as a linear programming problem.
- (12.3.4) is equivalent to the linear program

$$\begin{aligned} &\min_{V_i} \sum_{i=1}^{n} V_i \\ &s.t. \quad V_i \geq \pi(x_i, u) + \beta \sum_{j=1}^{n} q_{ij}(u) V_j, \ \forall i, u \in \mathcal{D}, \end{aligned} \tag{12.4.10}$$

- Computational considerations
  - (12.4.10) may be a large problem
  - Trick and Zin (1997) pursued an acceleration approach with success.
  - OR literature did not favor this approach, but recent work by Daniela Pucci de Farias and Ben van Roy has revived interest.

Continuous states: discretization

- Method:
  - "Replace" continuous X with a finite

$$X^* = \{x_i,\, i = 1, \cdots, n\} \subset X$$

  - Proceed with a finite-state method.
- Problems:
  - Sometimes need to alter space of controls to assure landing on an x in X.
  - A fine discretization often necessary to get accurate approximations

Continuous States: Linear-Quadratic Dynamic Programming

- Problem:

$$\max_{u_t} \sum_{t=0}^{T} \beta^t \left(\frac{1}{2} x_t^\top Q_t x_t + u_t^\top R_t x_t + \frac{1}{2} u_t^\top S_t u_t \right) + \frac{1}{2} x_{T+1}^\top W_{T+1} x_{T+1} \tag{12.6.1}$$

$$x_{t+1} = A_t x_t + B_t u_t,$$

- Bellman equation:

$$V(x,t) = \max_{u_t} \frac{1}{2} x^\top Q_t x + u_t^\top R_t x + \frac{1}{2} u_t^\top S_t u_t + \beta V(A_t x + B_t u_t, t+1). \tag{12.6.2}$$

Finite horizon

- Key fact: We know solution is quadratic, solve for the unknown coefficients
- The guess $V(x,t) = \frac{1}{2} x^\top W_{t+1} x$ implies f.o.c.

$$0 = S_t u_t + R_t x + \beta B_t^\top W_{t+1} (A_t x + B_t u_t),$$

  - F.o.c. implies the time t control law

$$\begin{aligned} u_t &= -(S_t + \beta B_t^\top W_{t+1} B_t)^{-1} (R_t + \beta B_t^\top W_{t+1} A_t) x \\ &\equiv U_t x. \end{aligned} \tag{12.6.3}$$

  - Substitution into Bellman implies *Riccati equation* for W_t:

$$W_t = Q_t + \beta A_t^\top W_{t+1} A_t + (\beta B_t^\top W_{t+1} A_t + R_t^\top) U_t. \tag{12.6.4}$$

  - Value function method iterates (12.6.4) beginning with known W_{T+1} matrix of coefficients.

Autonomous, Infinite-horizon case.

- Assume $R_t = R$, $Q_t = Q$, $S_t = S$, $A_t = A$, and $B_t = B$
- The guess $V(x) \equiv \frac{1}{2}x^\top W x$ implies the *algebraic Riccati equation*

$$\begin{aligned} W =& Q + \beta A^\top W A - (\beta B^\top W A + R^\top) \\ & \times (S + \beta B^\top W B)^{-1}(\beta B^\top W B + R^\top). \end{aligned} \tag{12.6.5}$$

- Two convergent procedures:
  - Value function iteration:

$$\begin{aligned} W_0 &: \text{a negative definite initial guess} \\ W_{k+1} =& Q + \beta A^\top W_k A - (\beta B^\top W_k A + R^\top) \\ & \times (S + \beta B^\top W_k B)^{-1}(\beta B^\top W_k B + R^\top). \end{aligned} \tag{12.6.6}$$

  - Policy function iteration:

$$\begin{aligned} W_0 &: \text{initial guess} \\ U_{i+1} =& -(S + \beta B^\top W_i B)^{-1}(R + \beta B^\top W_i A) : \text{optimal policy for } W_i \\ W_{i+1} =& \frac{\frac{1}{2}Q + \frac{1}{2}U_{i+1}^\top S U_{i+1} + U_{i+1}^\top R}{1 - \beta} : \text{value of } U_i \end{aligned}$$

Lessons

- We used a functional form to solve the dynamic programming problem
- We solve for unknown coefficients
- We did not restrict either the state or control set
- Can we do this in general?

Continuous Methods for Continuous-State Problems

- Basic Bellman equation:

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

  - Discretization essentially approximates V with a step function
  - Approximation theory provides better methods to approximate continuous functions.

- General Task
  - Find good approximation for V
  - Identify parameters

Parametric Approach: Approximating $V(x)$

- 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

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

  - polynomials with coefficients a and collocation points X
  - splines with coefficients a with uniform nodes X
  - rational function with parameters a and nodes X
  - neural network
  - specially designed functional forms

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

- Data for approximating $V(x)$

  - 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

- We review approximation methods.

Approximation Methods

- General Objective: Given data about a function $f(x)$ (which is difficult to compute) construct a simpler function $g(x)$ that approximates $f(x)$.
- Questions:
  - What data should be produced and used?
  - What family of "simpler" functions should be used?
  - What notion of approximation do we use?
  - How good can the approximation be?
  - How simple can a good approximation be?
- Comparisons with statistical regression
  - Both approximate an unknown function
  - Both use a finite amount of data
  - Statistical data is noisy; we assume here that data errors are small
  - Nature produces data for statistical analysis; we produce the data in function approximation
  - Our approximation methods are like experimental design with very small experimental error

Types of Approximation Methods

- Interpolation Approach: find a function from an n-dimensional family of functions which exactly fits 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
- 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.,
$$g = \arg\max_{g \in G} \|f - g\|^2$$

- Chebyshev polynomials - Example of orthogonal polynomials
  - $[a, b] = [-1, 1]$
  - $w(x) = \left(1 - x^2\right)^{-1/2}$
  - $T_n(x) = \cos(n \cos^{-1} x)$
  - Recurrence formula:

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

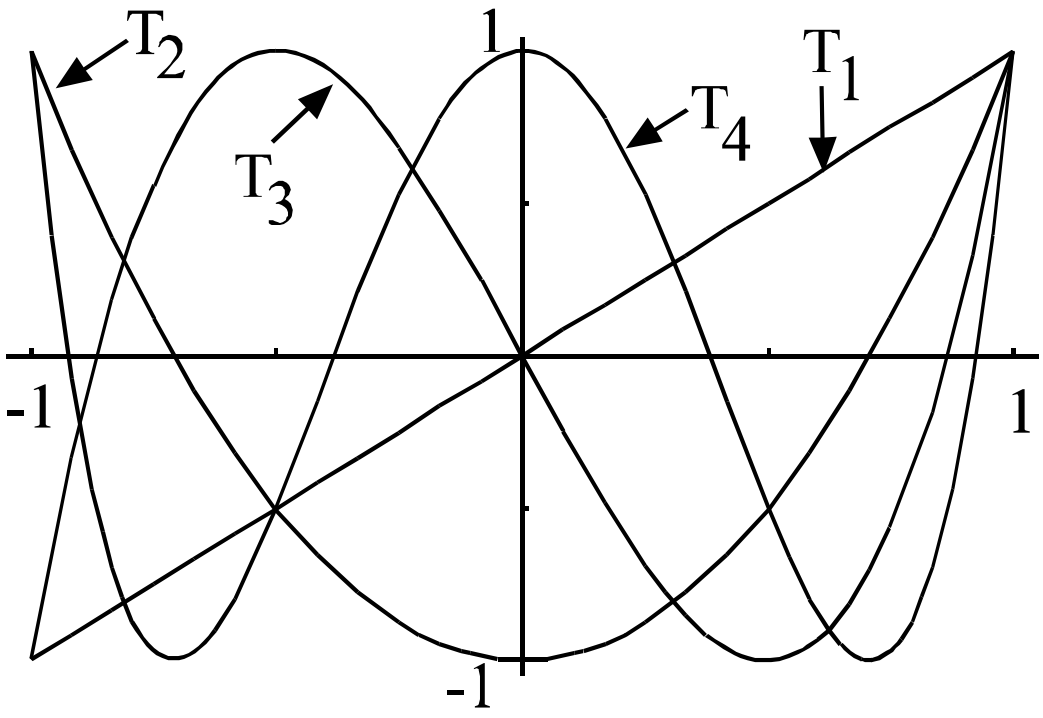


Figure 1:

- General intervals
  - Few problems have the specific intervals used in definition of Chebshev polynomials
  - Map compact interval $[a, b]$ 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 the Chebyshev polynomials adapted to $[a, b]$

Regression

- Data: $(x_i, y_i), i = 1, .., n.$
- Objective: Find $\beta \in R^m$, $m \leq n$, with $y_i \doteq f(x_i; \beta), i = 1, .., n.$
- 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 $[a, b]$

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 2 *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 3 *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

- B-Splines: A basis for splines
  - Put knots at $\{x_{-k}, \cdots, x_{-1}, x_0, \cdots, x_n\}$.
  - Order 1 splines: step function interpolation spanned by

$$B_i^0(x) = \begin{cases} 0, & x < x_i, \\ 1, & x_i \le x < x_{i+1}, \\ 0, & x_{i+1} \le x, \end{cases}$$

  - Order 2 splines: piecewise linear interpolation and are spanned by

$$B_i^1(x) = \begin{cases} 0\ , & x \le x_i \text{ or } x \ge x_{i+2}, \\ \frac{x - x_i}{x_{i+1} - x_i}\ , & x_i \le x \le x_{i+1}, \\ \frac{x_{i+2} - x}{x_{i+2} - x_{i+1}}\ , & x_{i+1} \le x \le x_{i+2}. \end{cases}$$

    The B_i^1-spline is the tent function with peak at x_{i+1} and is zero for $x \le x_i$ and $x \ge x_{i+2}$.
  - Both B^0 and B^1 splines form cardinal bases for interpolation at the x_i's.
  - Higher-order B-splines are defined by the recursive relation

$$B_i^k(x) = \left(\frac{x - x_i}{x_{i+k} - x_i} \right) B_i^{k-1}(x) + \left(\frac{x_{i+k+1} - x}{x_{i+k+1} - x_{i+1}} \right) B_{i+1}^{k-1}(x)$$

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

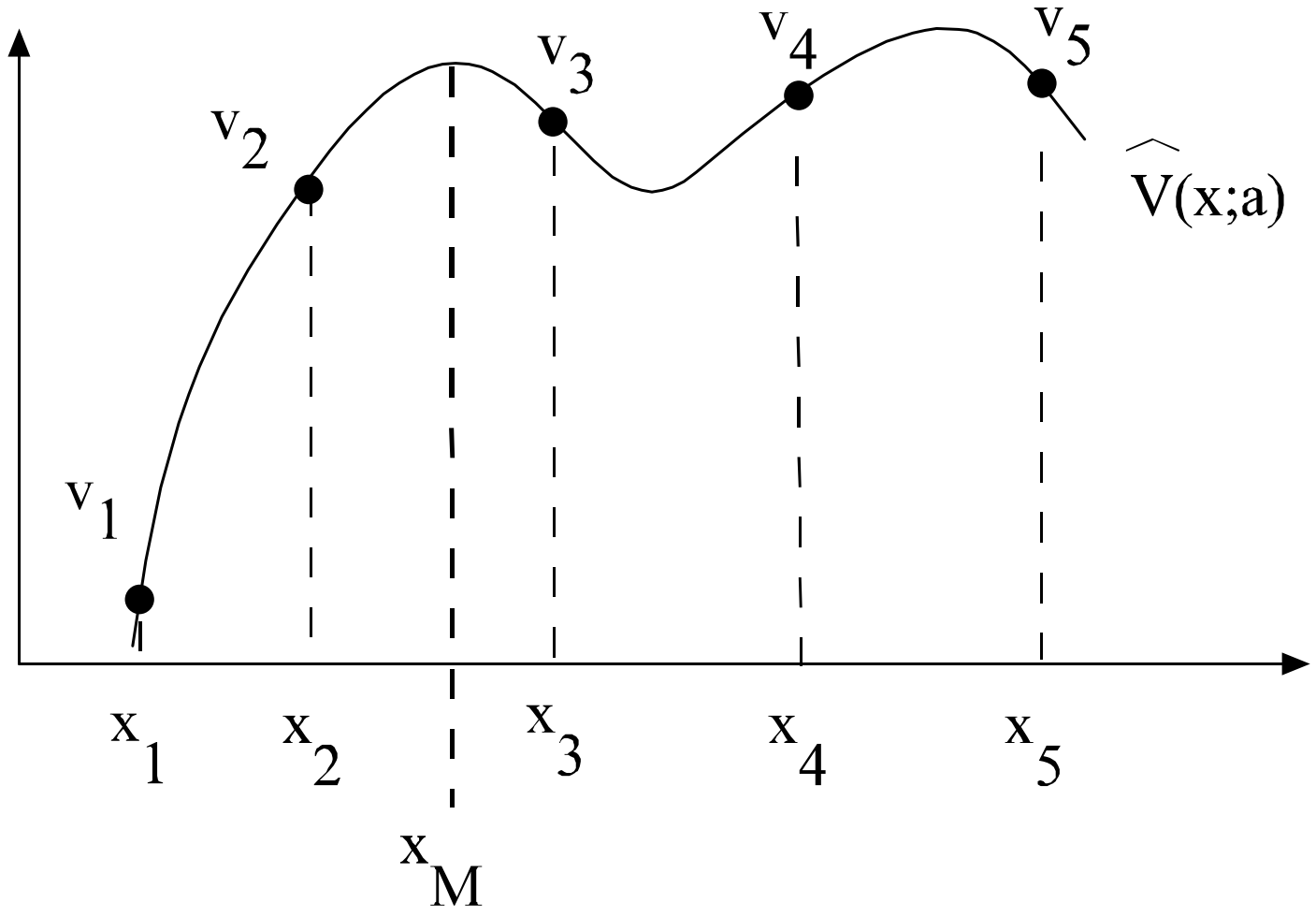


Figure 2:

- 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. Create quadratic spline with nodes at the x and z nodes to interpolate data and preserves shape.

- Many other procedures exist for one-dimensional problems
- Few procedures exist for two-dimensional problems
- Higher dimensions are difficult, but many questions are open.

- 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.
  - Good fits even for functions with discontinuous or large higher-order derivatives.
  - 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 $f : R^n \to R^m$ such that $z_i = f(x_i)$.
  - Task: Find combinations of interpolation nodes and spanning functions to produce a nonsingular (well-conditioned) interpolation matrix.

Tensor products

- 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

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

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

- In general, the complete set of polynomials of total degree k in n variables.

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

- 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

Parametric Approach: Approximating T - Expectations

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

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

- The definition includes an expectation

$$E\{V(x^+; a)|x_j, u)\} = \int \hat{V}(x^+; a) dF(x^+|x_j, u)$$

- How do we approximate the integral?

Integration

- Most integrals cannot be evaluated analytically
- We examine various integration formulas that can be used.

Newton-Cotes Formulas

- Trapezoid Rule: piecewise linear approximation
  - nodes: $x_j = a + (j - \frac{1}{2})h$, $j = 1, 2, \ldots, n$, $h = (b - a)/n$
  - for some $\xi \in [a, b]$

$$\int_a^b f(x)\, dx = \frac{h}{2} \left[f_0 + 2f_1 + \cdots + 2f_{n-1} + f_n\right] - \frac{h^2 (b - a)}{12} f''(\xi)$$

- Simpson's Rule: piecewise quadratic approximation

$$\int_a^b f(x)\, dx = \frac{h}{3} \left[f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 4f_{n-1} + f_n\right] - \frac{h^4 (b - a)}{180} f^{(4)}(\xi)$$

- Obscure rules for degree 3, 4, etc. approximations.

Gaussian Formulas

- All integration formulas are of form

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

  for some *quadrature nodes* $x_i \in [a, b]$ and *quadrature weights* ω_i.

  - Newton-Cotes 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 $2n - 1$

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 $\int_a^b f(x)\,dx$
  - Different range, no weight function
  - Linear change of variables $x = -1 + 2(y-a)(b-a)$
  - Multiply the integrand by $(1-x^2)^{1/2} \big/ (1-x^2)^{1/2}$.
  - C.O.V. formula

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

  - Gauss-Chebyshev quadrature produces

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

    where the x_i are Gauss-Chebyshev nodes over $[-1, 1]$.

Gauss-Hermite Quadrature

- Domain: $[-\infty, \infty]]$
- Weight: e^{-x^2}
- Formula:

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

  for some $\xi \in (-\infty, \infty)$.

- Example formulas

Table 7.4: Gauss – Hermite Quadrature

N	x_i	ω_i
3	0.1224744871(1)	0.2954089751
	0.0000000000	0.1181635900(1)
4	0.1650680123(1)	0.8131283544(−1)
	0.5246476232	0.8049140900
7	0.2651961356(1)	0.9717812450(−3)
	0.1673551628(1)	0.5451558281(−1)
	0.8162878828	0.4256072526
	0.0000000000	0.8102646175

- Normal Random Variables
  - Y is distributed $N(\mu, \sigma^2)$
  - Expectation is integration:

$$E\{f(Y)\} = (2\pi\sigma^2)^{-1/2} \int_{-\infty}^{\infty} f(y) e^{-\frac{(y-\mu)^2}{2\sigma^2}} \, dy$$

  - Use Gauss-Hermite quadrature
    - linear COV $x = (y - \mu)/\sqrt{2}\,\sigma$
    - COV formula:

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

    - COV quadrature formula:

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

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

Multidimensional Integration

- Many dynamic programming problems have several dimensions
- 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*

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

- 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.

- 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 $[-1,1]^d$

$$\int_{[-1,1]^d} f \doteq \omega \sum_{i=1}^{d} \left(f(ue^i) + f(-ue^i) \right)$$
$$u = \left(\frac{d}{3} \right)^{1/2}, \ \omega = \frac{2^{d-1}}{d}$$

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

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

    where

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

Parametric Approach: Approximating T - Maximization

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

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

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

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

  using some integration method.

$$E\{V(x^+; a)|x_j, u)\} \doteq \sum_\ell \omega_\ell \hat{V}(g(x_j, u, \varepsilon_\ell); a)$$

- We now come to the maximization step: for $x_i \in X$, evaluate

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

  - Use appropriate optimization method, depending on the smoothness of the maximand
  - Hot starts
  - Concave stopping rules

- When we have computed the v_i (and perhaps v_i') we execute the fitting step:

  - 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 the data
  - Methods: determined by $\hat{V}(x; a)$

Parametric Approach: Value Function Iteration

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

- 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 schemes allow us to use 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 for $\hat{V}(x;a)$, and choose the approximation 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

$$v_j = (T\hat{V}(\cdot;a^i))(x_j) \text{ for all } x_j \in X.$$

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

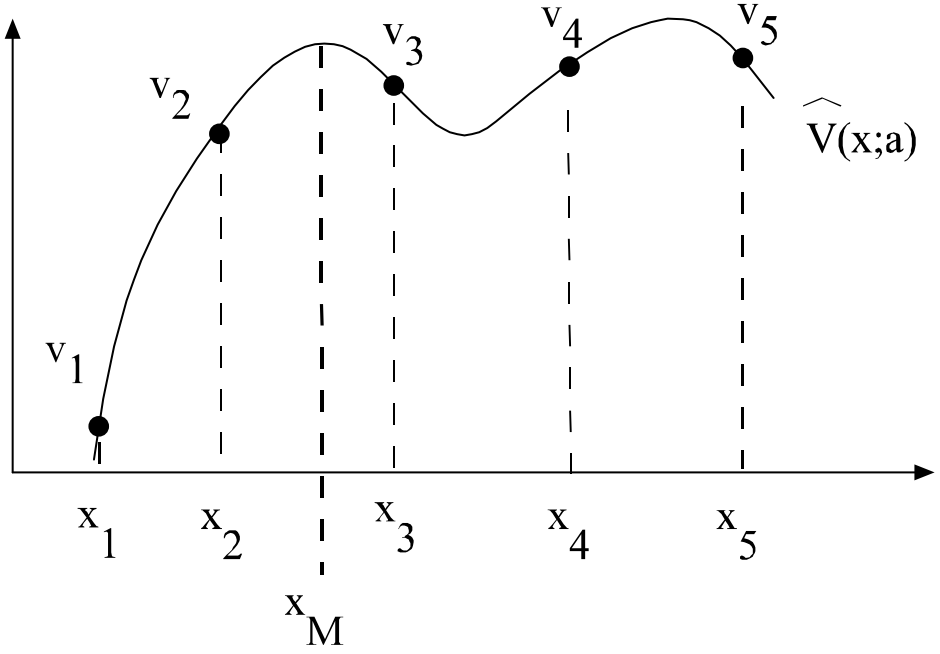


Figure 3:

  - Shape problems may become worse with value function iteration
  - Shape-preserving approximation will avoid these instabilities

Comparisons

We apply various methods to the deterministic growth model

	Relative L2 Errors over [0.7,1.3]					
N	(β, γ) :					
	(.95,-10.)	(.95,-2.)	(.95,-.5)	(.99,-10.)	(.99,-2.)	(.99,-.5)
	Discrete model					
12	7.6e-02	2.8e-03	5.3e-03	7.9e-01	1.8e-01	1.1e-02
1200	1.0e-04	2.1e-05	5.4e-05	2.9e-03	5.4e-03	1.3e-04
	Linear Interpolation					
4	7.9e-03	4.1e-03	2.4e-03	8.0e-03	4.1e-03	2.4e-03
12	1.5e-03	9.8e-04	5.6e-04	1.5e-03	1.0e-03	6.3e-04
120	1.1e-04	3.7e-05	1.3e-05	1.4e-04	8.4e-05	4.2e-05
	Cubic Spline					
4	6.6e-03	5.0e-04	1.3e-04	7.1e-03	5.7e-04	1.8e-04
12	8.7e-05	1.5e-06	1.8e-07	1.3e-04	4.9e-06	1.1e-06
40	7.2e-08	1.8e-08	5.5e-09	7.6e-07	8.8e-09	4.9e-09
120	5.3e-09	5.6e-10	1.3e-10	4.2e-07	4.1e-09	1.5e-09
	Polynomial (without slopes)					
4	DNC	5.4e-04	1.6e-04	1.4e-02	5.6e-04	1.7e-04
12	3.0e-07	2.0e-09	4.3e-10	5.8e-07	4.5e-09	1.5e-09
	Shape Preserving Quadratic Hermite Interpolation					
4	4.7e-04	1.5e-04	6.0e-05	5.0e-04	1.7e-04	7.3e-05
12	3.8e-05	1.1e-05	3.7e-06	5.9e-05	1.7e-05	6.3e-06
120	2.2e-07	1.7e-08	3.1e-09	4.0e-06	4.6e-07	5.9e-08
	Shape Preserving Quadratic Interpolation (ignoring slopes)					
4	1.1e-02	3.8e-03	1.2e-03	2.2e-02	7.3e-03	2.2e-03
12	6.7e-04	1.1e-04	3.1e-05	1.2e-03	2.1e-04	5.7e-05
120	2.5e-06	1.5e-07	2.2e-08	4.3e-06	8.5e-07	1.9e-07

General Parametric Approach: Policy Iteration

- Basic Bellman equation:

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

- Policy iteration:
  - Current guess: a finite-dimensional linear parameterization

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

  - Iteration: compute optimal policy today if $\hat{V}(x; a)$ is value tomorrow

$$U(x) = \pi_u(x_i, U(x), t) + \beta \frac{d}{du} \left(E \left\{ \hat{V}\left(x^+; a\right) |x, U(x)) \right\} \right)$$

    using some approximation scheme $\hat{U}(x; b)$
  - Compute the value function if the policy $\hat{U}(x; b)$ is used forever, which is solution to the linear integral equation

$$\hat{V}(x; a') = \pi(\hat{U}(x; b), x) + \beta \, E\{\hat{V}(x^+; a')|x, \hat{U}(x; b))\}$$

    that can be solved by a projection method

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