Computing for Economists

Dynamic Programming

Toren Fronsdal and Ruby Zhang

Harvard University

Introduction

- *Dynamic programming* (DP) first coined by Richard Bellman to describe his work on optimizing *multistage decision processes*
- He contributed the *Bellman equation*, which is a recursive formulation of the optimization problem
- Various contexts from economics, computer science, engineering, and bioinformatics encounter DP:
  - Dijkstra's shortest path algorithm (CS)
  - Job search model (labor)
  - Optimal savings problem (finance/macro)
  - Optimal firm investment (IO/finance/macro)
  - Industry life-cycle with entry and exit (IO/macro)

Resources

This module draws from:

- “Dynamic Programming” by Thomas Sargent and John Stachurski
- Jesús Fernández-Villaverde Courses
- Ken Judd Course and Book
- Matteo Courthoud Course

and will aim to avoid repeating material from ECON 2010C

Discrete vs. Continuous Time

DP can be in **discrete** or **continuous** time

- We will only focus on **discrete time** methods in this module
- Ito's Lemma in ECON 2010C $\rightarrow$ continuous time problems become a PDE
- Numerical methods to solve PDEs is very different, see slides for overview

Table of Contents

Basic Framework

Markov Decision Process (MDP)

Suppose an agent is faced with the problem of maximizing total expected rewards over time $t = \{0, 1, ..., T\}$ given discount factor β:

$$\mathbb{E}\left[\sum_{t=0}^{T} \beta^t r(X_t, A_t)\right]$$

where at a given time t:

1. The agent observes the current state X_t (initial state X_0 is given)
2. The agent chooses action/control variable $A_t \in \Omega(X_t)$
3. The agent receives reward $r(X_t, A_t)$
4. The agent transitions to state $X_{t+1} = P(.|X_t, A_t)$ for some stochastic kernel P (so $\sum_{X'} P(X'|X, A) = 1$)

MDP Overview

- With sufficiently large state space, *any* exponentially discounted problem with time-separable rewards can be framed as a MDP
  - Covers most discrete-time DP problems
  - Exceptions include time-varying discount rate such as $\beta_t = \frac{1}{1+r_t}$
- MDPs can vary along multiple dimensions:
  - **Finite** ($T < \infty$) or **infinite** ($T \to \infty$) horizon problem
  - **Discrete** or **continuous** state and action space
  - Let's first assume states and actions (X_t, A_t) are discrete
- MDPs are foundational to reinforcement learning

Bellman Equation and Operator

Instead of choosing a sequence of actions $\{A_t\}$, Bellman equations reframe the problem recursively in terms of **value function** $V(X)$:

$$V(X) = \max_{A \in \Omega(X)} \left\{ \underbrace{r(X,A)}_{\text{current reward}} + \overbrace{\beta \sum_{X'} V(X')P(X'|X,A)}^{\text{expected discounted flow of future value}} \right\}$$

We can define Bellman operator B which acts on function $V(X)$:

$$BV(X) = \max_{A \in \Omega(X)} \left\{ r(X,A) + \beta \sum_{X'} V(X')P(X'|X,A) \right\}$$

B is a contraction mapping under Blackwell's sufficient conditions:

Optimal value function $V^* = BV^*$ is a fixed point

This is also a **non-linear** system of equations due to the max operator

Policy Functions

- Let $\alpha(X)$ denote the **policy function** given the state
  - Optimal policy function $\alpha^*(X)$ maximizes $V(X)$:

$$\alpha^*(X) = argmax_{A \in \Omega(X)} \left\{ r(X, A) + \beta \sum_{X'} V(X') P(X'|X, A) \right\}$$

- Define V_α as the value function under *some* policy $\alpha(X)$:

$$V_\alpha(X) = r(X, \alpha(X)) + \beta \sum_{X'} V_\alpha(X') P(X'|X, \alpha(X))$$

- Given a specific policy function $\alpha(X)$, computing V_α is a **linear** system of equations:
  - Let vector $r_\alpha \equiv r(X, \alpha(X))$
  - Let matrix $P_\alpha \equiv P(X'|X, \alpha(X))$
  - Then $V_\alpha = (I - \beta P_\alpha)^{-1} r_\alpha$

Numerical Solutions

Value Function Iteration (VFI)

The canonical DP solution is via VFI:

1. Start with guess $V^0(X)$ (length $|\{X_i\}|$ for all possible states X_i)
2. Iterate at each step to get the next value function:

$$V^{k+1}(X) = \max_{A \in \Omega(X)} \left\{ r(X, A) + \beta \sum_{X'} V^k(X') P(X'|X, A) \right\}$$

3. Repeat until $\|V^{k+1}(X) - V^k(X)\| < \epsilon$ for some threshold ϵ

Note that the calculation of $V^{k+1}(X_i)$ is independent across possible states $\{X_i\}$, thus it is **parallelizable**

VFI in Finite-Horizon Problems

- Backward induction based on terminal condition V^T:

$$V^T(X) = \max_{A \in \Omega(X)} r(X, A)$$

- Recursively solve α^t and V^t since V^{t+1} known:

$$\alpha^t(X) = argmax_{A \in \Omega(X)} \left\{ r(X, A) + \beta \sum_{X'} V^{t+1}(X') P(X'|X, A) \right\}$$

$$V^t(X) = r(X, \alpha^t(X)) + \beta \sum_{X'} V^{t+1}(X') P(X'|X, \alpha^t(X))$$

- VFI is the only possible procedure for finite horizon case
  - $V(X)$ is dependent on t

- In the infinite-horizon case, only state matters

Computational Breakdown

There are two expensive operations at each iterative step:

1. Maximizing the value function by choosing the optimal policy
    - Brute force calculation of value function over all possible actions
    - Optimization or root-finding of FOCs using quasi-Newtonian methods (especially if concave and smooth)
        - Automatic differentiation is fast and parallelizable
    - If the problem is concave, can try until $A_{i+1} < A_i$
    - Early iterations **do not** require accuracy $\rightarrow$ try subset of $\{A_i\}$ or loose convergence criterion in maximization
2. Integrating over all possible future states
    - Numerical integration methods: quadrature, monte carlo
    - Approximation methods for complicated integrals

Gauss-Seidel Algorithm

Speed up VFI with Gauss-Seidel algorithm:

- Successively substitute each state $V^{k+1}(X_i)$ to calculate $\beta \sum_{X'} V^k(X')P(X'|X,A)$:

$$\beta \sum_{j<i} V^{k+1}(X_j)P(X_j|X_i,A) + \beta \sum_{j\geq i} V^k(X_j)P(X_j|X_i,A)$$

- Con: Cannot exploit parallelizability

- Pro: Quick convergence if states are highly coupled
  - The order of state calculations matters
  - More absorbing states should be calculated first since a "terminal" value function is more accurate

Howard Policy Iteration (HPI)

Idea: Instead of iterating on the value function, alternate between policy and value functions

1. Start with (the better) guess $V^0(X)$ or $\alpha^0(X)$ (which is of length $|\{X_i\}|$ for all possible states)
2. At each step, compute the optimal policy:

$$\alpha^{k+1}(X) = \text{argmax}_{A \in \Omega(X)} \left\{ r(X, A) + \beta \sum_{X'} V^k(X') P(X'|X, A) \right\}$$

3. Compute the value function taking the optimal policy as given (linear system):

$$V^{k+1}(X) = r(X, \alpha^{k+1}(X)) + \beta \sum_{X'} V^{k+1}(X') P(X'|X, \alpha^{k+1}(X))$$

$$\equiv$$

$$V^{k+1} = (I - \beta P_{\alpha^{k+1}})^{-1} r_{\alpha^{k+1}}$$

4. Repeat until $\|V^{k+1}(X) - V^k(X)\| < \epsilon$ for some threshold ϵ

VFI vs. HPI

VFI is dominated by HPI in terms of speed:

- VFI is a fixed-point algorithm and converges linearly at rate β (larger β means slower convergence)
- HPI is a Newtonian algorithm (i.e. solving $f(x) - x = 0$ instead of $f(x) = x$) and converges quadratically

VFI and HPI differ by how many iterations of $\alpha^k(X)$ is used:

- VFI calculates V^{k+1} assuming α^{k+1} is used for one period
- HPI calculates V^{k+1} assuming α^{k+1} is used for ∞ periods

However, what if $(I - \beta P_\alpha)$ is hard to invert?

Optimistic Policy Iteration (OPI)

Idea: Combine VFI and HPI by iterating with α for $1 < m < \infty$ steps

1. Start with guess $V^0(X)$ or $\alpha^0(X)$ (which is of length $|\{X_i\}|$ for all possible states)
2. At each step, compute the optimal policy:

$$\alpha^{k+1}(X) = \text{argmax}_{A \in \Omega(X)} \left\{ r(X, A) + \beta \sum_{X'} V^k(X') P(X'|X, A) \right\}$$

3. Approximate $V^{k+1}(X)$ by iterating with $\alpha^{k+1}(X)$ for m steps:

$$V^{k+1}(X) = B_\alpha^m V^k(X)$$

4. Repeat until $\|V^{k+1}(X) - V^k(X)\| < \epsilon$ for some threshold ϵ

Without parallelization, OPI can be more efficient (no inversion and use the policy function)

Linear Programming (LP)

For discrete states and actions, frame as linear programming problem:

$$\min_{V_i} \sum_{i=1}^{n} V_i \text{ s.t. } V_i \geq r(X_i, A_j) + \beta \sum_{X'} V(X') P(X'|X_i, A_j) \quad \forall i, j$$

- Pro: no need for iterative algorithms, well-established linear programming methods (simplex method, interior point methods)
- Con: discrete can still mean *very large*
- Not commonly used in economics, see engineering literature (de Farias and van Roy, 2003)

Summary of Methods

Method	Pros	Cons
VFI	Simple, easy to implement, converges reliably and exploit parallelizability	Can be slow for high-dimensional and large state spaces, converge linearly
Gauss-Seidel	Improvement on VFI especially for highly-coupled states	More complex to implement and may still be slow, not parallelizable
HPI	More efficient than VFI (quadratic convergence), especially when the policy is stable	Invert matrix to compute value function at each step (especially if complicated transitions)
OPI	Combines advantages of VFI and PFI, good for more complex problems	Less direct than VFI
LP	Direct and non-iterative solution methods	Only for discrete actions and states with lower-dimensions

Discretization

Continuous State Space

- In IO and labor settings, discrete states arise naturally:
  - Quality ladder, number of competitors
- However, especially in **macro**, the state (and action) space is naturally continuous
  - Wealth, capital accumulation

Two common approaches:

1. Discretize the state space with grids
   - Lower dimensional, bounded state-space that's well approximated
2. Numerically approximate a continuous function
   - Complex, unbounded state space where action may be continuous (for maximization step)

State Space Discretization

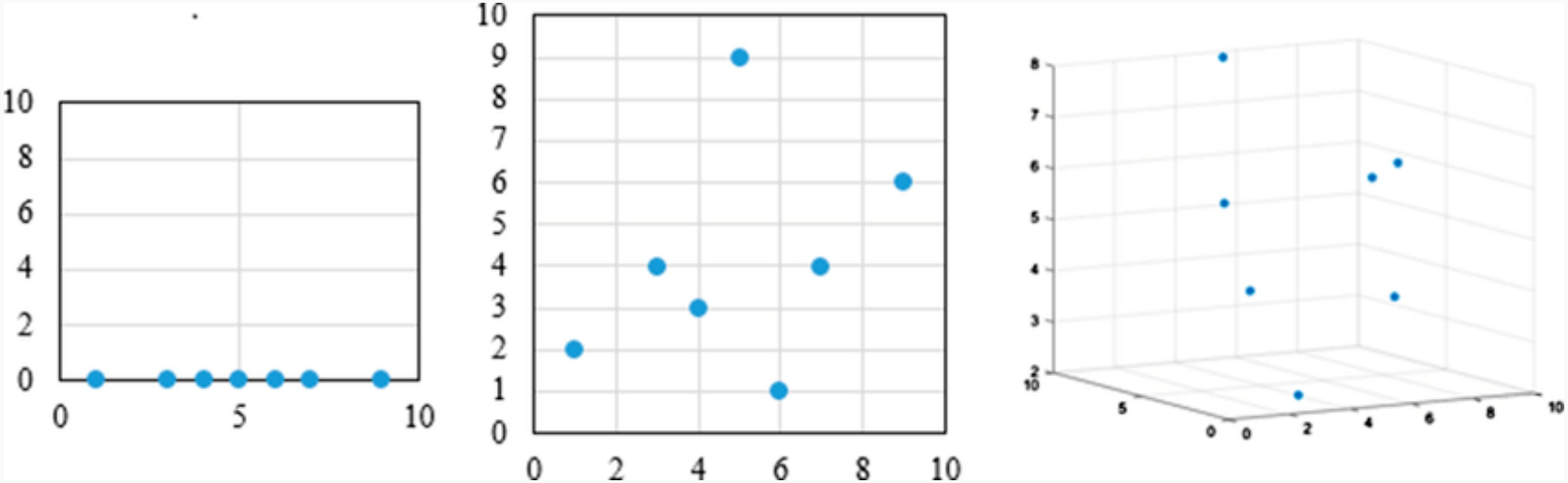
Goal: Convert functions of continuous states to finite vectors:

$$\beta \int V(X')P(dX'|X,A) \to \beta \sum_{X'} \hat{V}(X')P(X'|X,A)$$

- First need to select points $\{X_i\}$ then calculate $P(.)$:
  - Uniform grid is most direct (especially for bounded state spaces)
  - Intuitively, $V(.)$ with more curvature $\to$ more grid points
    - Tauchen's method
    - Stochastic grid
    - Endogenous grid
- Often, interpolate for states outside of grid
- Aside: limit your state space if possible due to curse of dimensionality!

Curse of Dimensionality

Necessary amount of points needed increases exponentially: N^d



Source

Tauchen's Method

For state variables that evolve via AR(1), let Y^t be income at time t:

$$Y^{t+1} = \rho Y^t + \epsilon^{t+1}, \quad \epsilon^{t+1} \sim \mathcal{N}(0, \sigma^2)$$

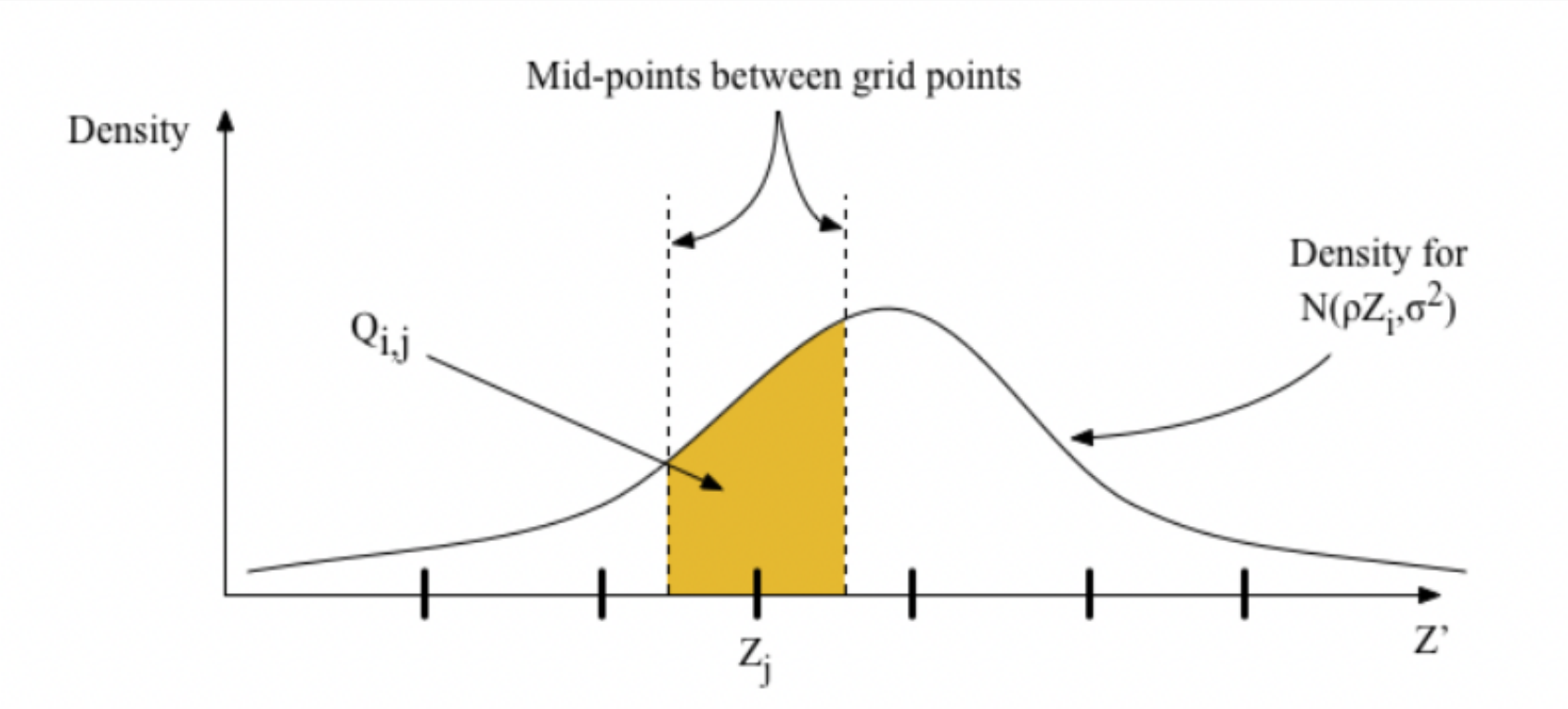
1. Select number of grid points N and some number m (e.g. 3) to get fixed, uniform grid:

$$[Y_1, ..., Y_N] \text{ where } Y_1 = -m\sigma, Y_N = m\sigma$$

2. Compute transition probabilities $P(Y^{t+1} = Y_j | Y^t = Y_i)$ for each pair (Y_i, Y_j) using density of $\mathcal{N}(\rho Y_i, \sigma^2)$ in the interval around Y_j

Faster and related method is Rouwenhorst

Tauchen's Method



Source

Stochastic Grid

- Fixed grid can lead to underexploration of some parts of state space

- Rust (1995) proposed a stochastic grid to get (1) exploration (2) computational feasibility
  1. At every iteration, randomly generate a grid of state variables
     - If applicable, use probability distribution of state variables
  2. Calculate value and policy functions only at the generated grid
  3. For all non-grid values of X, approximate $V(X)$ and $\alpha(X)$ using linear interpolation or splines

Endogenous Grid

Idea: Carroll (2006) proposed starting with an exogenous grid on choices, and get the state that would induce the choice

- Reduces computational burden of root-finding from FOC's at each state grid point
- Depending on the problem at hand, it's easier to invert FOC's to construct middle-of-period values
- Can deal with kinks in the policy function, will naturally locate grid where more likely states are found
- Useful for settings with endogenously determined state transitions (e.g. life-cycle savings model)
  - Less applicable to stochastic state transitions

High-Dimensional Dynamic Programming

- As opposed to a representative agent, what if there are many?
  - Countries, firms, households, networks, etc.
  - Agents must keep track of own and others' actions $\rightarrow$ curse of dimensionality
- Let x represent the agent's own state vector, and X represent other agents' states (with A as own action)
- If the agents are **symmetric** so the dynamic problem is **permutation-invariant**, then $\exists$ functions ρ and ϕ such that

$$r(x, X, A) = \rho\left(x, \frac{1}{N}\sum_{i=1}^{N}\phi(X_i), A\right)$$

- Fernández-Villaverde (2020) uses neural nets to approximate $\rho(.)$ and $\phi(.)$ and compare with Euler errors

Approximation Methods

Smooth Approximations

- Discretization $\equiv$ approximate $V(X)$ with step function
  - Improved with linear interpolation to capture states outside grid
- **Idea:** At every iteration, use a grid of points and coefficients γ to approximate a continuous function $V(X)$ with $\hat{V}(X;\gamma)$
  - For a smooth $\hat{V}(X;\gamma)$, can use quasi-Newton methods to maximize
  - Also called *projection methods* using polynomial basis
- Many possible approximation methods for continuous functions:[1]
  - Polynomials
  - Splines

[1] See Ken Judd slides on approximation methods.

VFI with Parametric Approximation

Idea: Approximate $\hat{V}(X;\gamma)$ at every iteration using grid of states

1. Choose functional form of $\hat{V}(X;\gamma)$ and grid $\{X_i\}$[2]
2. Calculate $\hat{V}(X;\gamma^0)$ with initial guess of γ^0
3. Compute data points at step k for each $\{X_i\}$:

$$V_i = \max_{A\in\Omega(X)} \left\{ r(X_i, A) + \beta \int \hat{V}(X';\gamma^k) P(dX'|X_i, A) \right\}$$

4. Calculate γ^{k+1} using data points (X_i, V_i) (i.e. Lagrange data) via least squares:

$$\text{argmin}_\gamma \|V_i - \hat{V}(X_i;\gamma)\|^2$$

5. Repeat until $\|\hat{V}(X;\gamma^{k+1}) - \hat{V}(X;\gamma^k)\| < \epsilon$ for some threshold ϵ

[2] Grid is guided by the approximation method chosen and can be adjusted during iterations.

Lagrange Polynomials

- By Stone-Weierstrass, every continuous function defined on $[a, b]$ can be approximated arbitrarily closely by a polynomial
- Lagrange polynomial interpolation for n data points (X_i, Y_i):

$$\hat{V}(X; \gamma) = \sum_{j=0}^{n-1} \gamma_j X^j$$

  - Pro: simple and direct to implement
  - Con: numerically unstable as number of nodes and polynomial degree grow, oscillation between interpolation points

Chebyshev Polynomials

- Chebyshev polynomial of the first kind (on interval $[-1, 1]$, can be generalized to any interval, see Ken Judd slides for details):

$$\hat{V}(X;\gamma) = \sum_{j=0}^{m} \gamma_j cos(j cos^{-1}(X))$$

  - Pro: numerical stability
  - Con: interpolation at specific Chebyshev nodes (calculated)

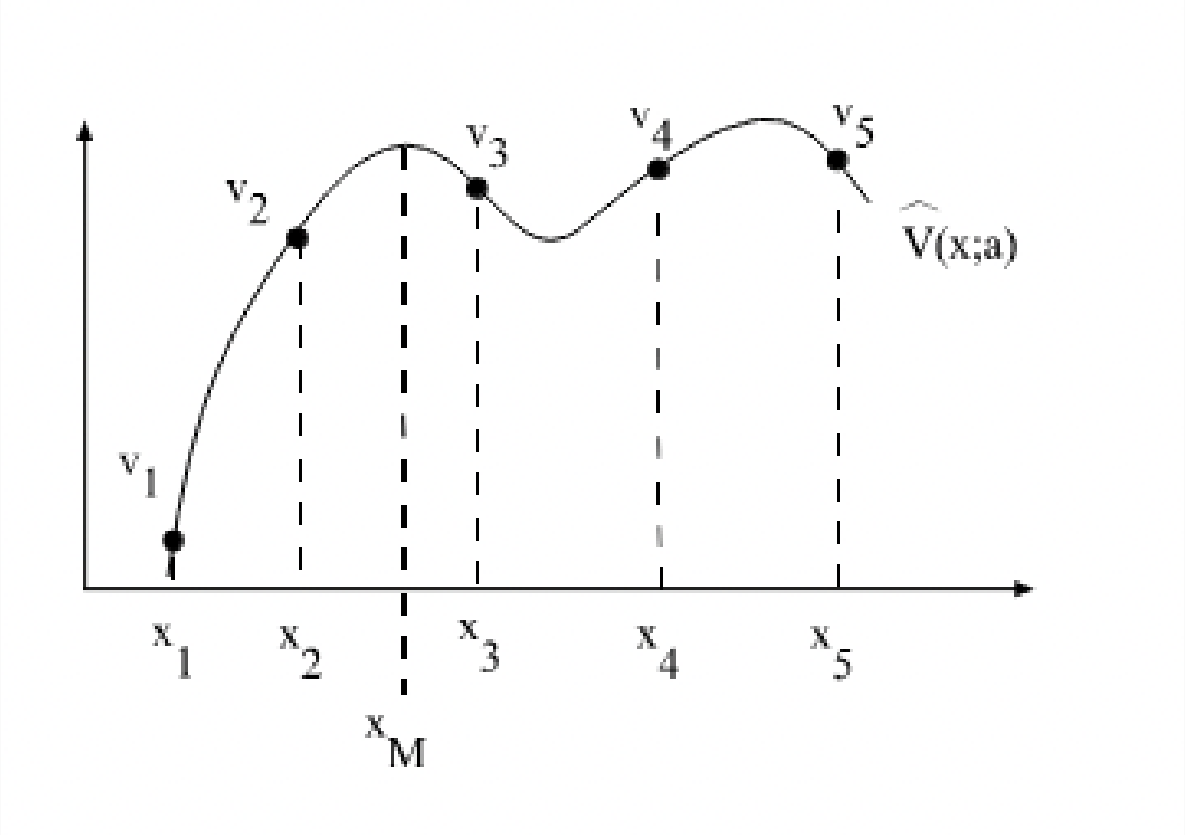
- Chebyshev polynomials are the most preferred basis for interpolation given stability even in complex problems
  - Endogenously chosen nodes can provide more stability

Splines

- Spline of order m has a degree $m-1$ polynomial for every subinterval $[X_i, X_{i+1}]$
- Spline of order 2 is equivalent to linear interpolation
- Spline of order 3 would have $4n$ unknowns for n data points
- However, splines suffer from the same instability problem as Lagrange polynomials
- Hermite splines also utilize data on $Y'(X_i)$ to match the first derivative at nodes

Shape Preservation

Concave data points can have non-concave approximations due to fluctuations between data points $\rightarrow$ instability



Source

Shape Preservation Methods

- If $V(X)$ is known to be concave or monotone, then so should approximation
- Linear interpolation preserves concavity but kinks make maximization step difficult
- Chebyshev's polynomials:
  - Add extra constraints for second derivative of $\hat{V}(X)$ for coefficient calculation
- Focus on Schumaker's splines and Hermite interpolation
  - Combine level and derivative information in approximation

Schumaker's Interpolation (1983)

Formula for shape-preserving splines (i.e. monotonicity and convexity) from Schumaker (1983)

- Take data points $(X_i, Y_i, Y'(X_i))$
- Intermediate nodes added at $Z_i \in [X_i, X_{i+1}]$ via Schumaker's formula
- Compute quadratic splines which satisfy $Y'(X_i)$ at X_i, Z_i nodes

Schumaker's formulas specify constraints on first and second derivatives between nodes depending on data shape

Multidimensional Approximation

- Multidimensional extensions are harder, usually involve tensor products to form basis
- Recall univariate basis of functions: $\sum_{j=0}^{n-1} \gamma_j \phi_j(X)$
- Suppose $X \in \mathbb{R}^2$ so $X = (X_1, X_2)$, then the approximation is:

$$\hat{V}(X; \gamma) = \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} \gamma_{kj} \phi_j(X_1) \phi_k(X_2)$$

- As dimensions grow to p the number of parameters becomes n^p
- Other complete polynomials such as Chebyshev are useful
  - Grows at slower pace than n^p since smaller univariate basis

DP Summary

Considerations when solving discrete-time dynamic programming:

1. Is the time horizon finite or infinite?
   - Finite: backward induction with VFI
   - Infinite: method depending on state space
2. Is the state/action space discrete or continuous?
   - Discrete: select appropriate method, some form of policy iteration for efficiency if high dimensional
   - Continuous:
     - Discretize state space if lower dimensional and finite $\rightarrow$ methods with discrete state/action space
     - Approximation if higher dimensional, unbounded and smooth function $\rightarrow$ iterative method with parametric estimation

Optional: Perturbation Methods

Local Methods

- Full solution methods characterize entire policy/value functions

Perturbation methods:

- **Goal:** locally approximate policy function $\alpha(X)$ around a **known** steady-state
  - Useful for both deterministic and stochastic model
- Use Taylor's expansion $\rightarrow \alpha(X)$ valid within a radius of convergence
  - Can perturb on equilibrium conditions or value function, we focus on former
- Euler equation partially characterize policy function and can be used to evaluate accuracy

Used in macro for solving DSGE problems

Euler Equation and Steady-State

Suppose we have the optimal growth problem:

$$\max_{C_t} \sum_{t=0}^{T} \beta^t u(C_t) \text{ s.t. } K_{t+1} + C_t = F(K_t)$$

- State: K_t
- Action/Policy function: $C_t = C(K_t)$

Euler equation derived from FOCs which link intertemporal controls:

$$u'(C_t) = \beta u'(C_{t+1}) F'(K_{t+1})$$

Steady-state (K^*, C^*) uniquely characterized by equilibrium conditions:

$$u'(C^*) = \beta u'(C^*) F'(K^*)$$
$$F(K^*) = K^* + C^*$$

Taylor Expansion

Approximate policy function $C(K)$ with a Taylor expansion around steady-state K^*:

$$C(K) \approx C(K^*) + C'(K^*)(K - K^*) + \frac{1}{2}C''(K^*)(K - K^*)^2$$

- If we knew $C'(K^*)$ and $C''(K^*)$, pin down policy function $C(K)$ locally

Idea: Take derivative of (substituted) Euler equation w.r.t. state and solve for $C'(K^*)$:

$$\text{Euler Equation: } u'(C(K_t)) = \beta u'(C(F(K_t) - C(K_t)))F'(F(K_t) - C(K_t))$$

$$\begin{aligned}\text{Derivative at } K^*\text{: } u''(C^*)C'(K^*) &= \beta u''(C^*)C'(K^*)[F'(K^*) - C'(K^*)] \\ &\quad + \beta u'(C^*)F''(K^*)[F'(K^*) - C'(K^*)]\end{aligned}$$

To calculate $C''(K^*)$, take second derivative of Euler equation to get linear equation in $C''(K^*)$ and known $C'(K^*)$

Perturbation Method

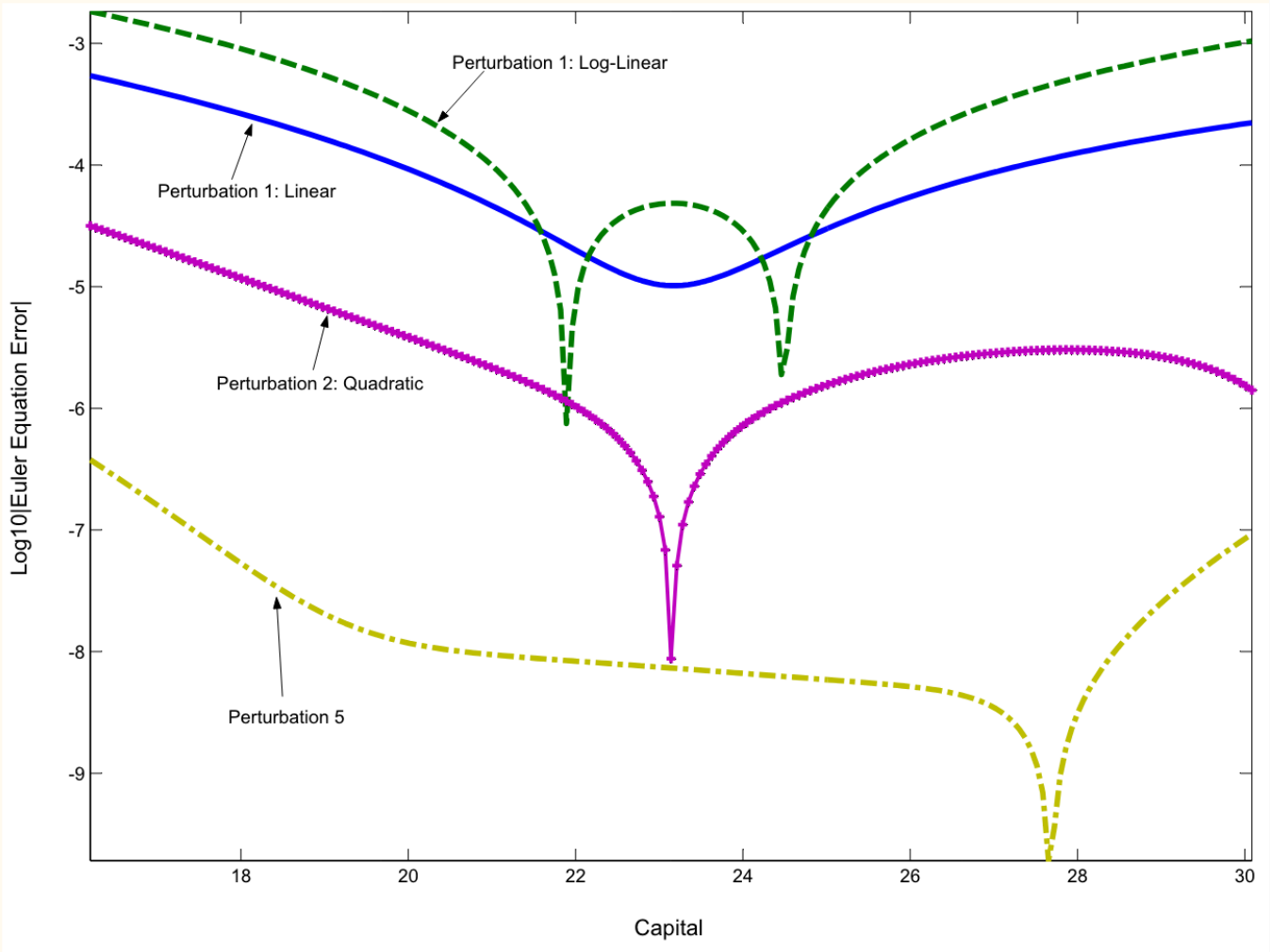
- For stochastic growth in $F(K_t)$, expand state space and assume zero in steady-state, along with perturbation parameter[3]

- Perturbation methods:[4]
  1. Calculate steady-state
  2. Taylor expansion of policy function around steady-state
  3. Differentiate equilibrium conditions (i.e. Euler equation) n-times
     $\rightarrow$ system of equations for n^{th}-order derivatives
  4. Substitute higher-order derivatives into Taylor expansion

- Euler equation can help assess quality of approximation
  - Take difference of two sides of equation (Euler errors)
  - Even for full-solution methods, can help with grid adjustment
  - If state cyclical, simplifies problem via multiperiod Euler equation

[3] See slides for stochastic growth in $F(K_t)$
[4] See slides for general solution methods

Euler Equation Errors



Source

Optional: Structural Estimation

What is Structural Estimation?

- Aside from macro, DP used in dynamic structural models
- Structural DP comprises of:
  - **Data** on observable state variables X_{it} and action variable A_{it}
  - Some **unobservable state** variable ϵ_{it} with distributional assumptions
    - **Uncertainty is fundamental** – no data fits the model perfectly!
    - E.g. Type I EV in discrete choice models
  - **Unknown parameters** θ part of the reward function $r(X, A; \theta)$ and thus value function $V(X; \theta)$
- Goal: optimizing the DP problem to **identify** θ
  - Solving DP problem is inner loop of the optimization over θ
  - Overall objective function is usually MLE or GMM
- Aside: GMM used by Hansen and Singleton (1982) to estimate parameters with Euler equations as moments

Unobservable States

How do we deal with the unobservable state ϵ_t?

- For example: productivity shock, private cost shock, shock to entry cost

Assuming IID and distributional form, integrate out ϵ_t

- Modified value function (expected value function):

$$\overline{V}(X_t) = \mathbb{E}_\epsilon\{V(X_t, \epsilon_t)\}$$

- Expected continuation value is expectation over X_t and ϵ_t

Some forms of ϵ distribution have tricks

- Suppose ϵ_t is additive in $r(X_t, A_t)$ and Type I EV

$$\mathbb{E}_\epsilon\left[\max_n\left(\{\delta_n + \epsilon_n\}_{n=1}^N\right)\right] = 0.57722 + \ln\left(\sum_{n=1}^N e^{\delta_n}\right)$$

Single-Agent Dynamics

- Suppose a *single* agent (e.g. firm) is making decisions A at each t that influence the state evolution X_{t+1}
- Econometrician observe states X_t and actions A_t
- Value function $V(X_t; \theta)$ determines the policy $\alpha(X_t; \theta)$ which determines choice probabilities $P(A_t|X_t; \theta)$
- Likelihood function is given by: $\mathcal{L}(\theta, V(\theta); X)$
- The classic DP approach follows nested fixed point:
  1. Taking θ as given, inner loop performs VFI to compute $V(\theta)$ and get choice probabilities $P(A_t|X_t; \theta)$
  2. Outer loop searches over the space of θ to maximize likelihood of calculated choice probabilities using data on choices
- The inner loop tolerance should be very tight (at least 10^{-14}) to prevent errors bubbling through

Alternative Solution Methods

For every possible θ, VFI is performed which can be very slow

- Alternatives in structural estimation try to get around VFI
  - Remember, there's data in this case!

MPEC: Mathematical Programming with Equilibrium Conditions

- **Idea**: use Bellman equations to formulate problem as constrained nonlinear optimization

$$\max_{\theta, V(\theta)} \mathcal{L}(\theta, V(\theta); X)$$

$$\text{s.t. } V(\theta) - BV(\theta) = 0$$

- Depending on the problem, constrained nonlinear optimization may still be computationally difficult

Dynamic Games (Multi-Agent Dynamics)

- In industry dynamics, firms respond to each other's actions strategically
- Discrete time stochastic game is characterized by Nash equilibrium $\rightarrow$ set of nonlinear equations
- Even though problem formulated as DP, there is no contraction factor so **DP methods do not apply**!
- Dynamic games are difficult and plagued by multiple equilibria (so not a fixed point)
- Given optimal policy and transitions, **forward simulation** of value functions will play key role

...not covered here!