Quantitative Macroeconomics w/ AI and ML
Lec. 7: Numerical Solution Methods for Dynamic Models

Zhigang Feng

Dec., 2025

Foundations of Dynamic Optimization

Dynamic Optimization in Macroeconomics

- Core problems: optimal growth, consumption-saving, asset pricing, labor supply, investment, etc.
- Common structure: an agent chooses actions $\{a_t\}_{t=0}^T$ to maximize

$$\max \mathbb{E}\left\{\sum_{t=0}^{T} \beta^t u(s_t, a_t) \mid s_0\right\}$$

  subject to constraints linking states and actions over time.
- With uncertainty, future states $s_1, s_2, \ldots$ are unknown at time 0.
- So we cannot choose a sequence of numbers; we must choose **contingent plans** $\alpha_t : S \to A$ specifying what action to take in each possible state.
- The **sequential problem**: optimize over function sequences $\{\alpha_t\}$, not number sequences.

Why Infinite Horizons?

- Finite horizon $T < \infty$: solution depends on t (non-stationary).
- Infinite horizon $T = \infty$:
  - Stationarity: optimal policy $\alpha(s)$ does not depend on calendar time.
  - Avoids arbitrary terminal conditions (e.g., "consume everything at T").
  - Natural for macroeconomic questions: long-run growth, business cycles, asset pricing.
- Practical: even for "finite-lived" agents, overlapping generations or dynastic models effectively create infinite horizons.

Why Markov Structure?

- General problem: action may depend on entire history $a_t = \alpha_t(s_t, s_{t-1}, a_{t-1}, \ldots)$.
- **Markov property**: the state s_t is a *sufficient statistic* for the future.

$$p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = p(s_{t+1}|s_t, a_t)$$

- Combined with additive separability of preferences: optimal policy is Markovian, $a_t = \alpha(s_t)$.
- This reduces the problem from searching over history-dependent strategies to searching over functions of the current state alone.

The Markov Decision Process (MDP)

- An MDP is defined by:
    - State space S, action space A, feasibility correspondence $A(s) \subseteq A$
    - Transition probability $p(s'|s,a)$
    - Per-period payoff $u(s,a)$, discount factor $\beta \in (0,1)$
- The sequential problem:

$$V(s_0) = \max_{\{\alpha_t\}} \mathbb{E}\left\{ \sum_{t=0}^{\infty} \beta^t u(s_t, \alpha_t(s_t)) \mid s_0 \right\}$$

- Difficult: optimization over function sequences, nested expectations.

From Sequential to Recursive: The Key Insight

- **Dynamic Programming** transforms the sequential problem into a recursive one.
- The value function $V(s)$ summarizes the continuation value from state s.
- **Bellman equation**:
$$V(s) = \max_{a \in A(s)} \left[u(s,a) + \beta \int V(s') p(ds'|s,a) \right]$$
- Maps a multi-period stochastic problem into a sequence of static optimization problems.
- The optimal policy $\alpha(s)$ achieves the maximum for each s.

Principle of Optimality

- **Bellman's Principle of Optimality**: An optimal policy has the property that, regardless of initial state and initial decision, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.
- This guarantees **equivalence**: the solution to the recursive Bellman equation yields the same optimum as the sequential problem.
- Key requirements: time-separable preferences, Markov transitions, stationarity.
- When these hold, we can solve the "easy" recursive problem instead of the "hard" sequential one.

Existence and Uniqueness: Contraction Mapping

- Define the **Bellman operator** B on the space of bounded continuous functions:

$$(BV)(s) = \max_{a \in A(s)} \left[u(s,a) + \beta \int V(s') p(ds'|s,a) \right]$$

- **Blackwell's sufficient conditions**: B is a contraction if it satisfies:
  - Monotonicity: $V \leq W \Rightarrow BV \leq BW$
  - Discounting: $B(V + c) \leq BV + \beta c$ for $c \geq 0$
- **Contraction Mapping Theorem**: B has a unique fixed point V^*, and $B^n V \to V^*$ for any initial V.
- This guarantees existence, uniqueness, and provides a computational algorithm.

Theorem of the Maximum

- Contraction mapping gives existence/uniqueness. But is the solution “nice”?
- **Theorem of the Maximum** (Berge): If $u(s, a)$ is continuous and $A(s)$ is a continuous correspondence, then:
  - The value function $V^*(s) = \max_{a \in A(s)} f(s, a)$ is continuous in s.
  - The policy correspondence $\alpha^*(s) = \arg\max_{a \in A(s)} f(s, a)$ is upper hemicontinuous.
- Applied to DP: ensures V is continuous and measurable selections of $\alpha(s)$ exist.
- Essential for justifying numerical approximation methods.

Properties of the Value Function

- Under standard conditions (concave u, convex $A(s)$, appropriate p):
  - V is **continuous** (Theorem of the Maximum)
  - V is **strictly concave** (concavity preserved under B)
  - V is **differentiable** (under stronger conditions; Benveniste-Scheinkman)
- Policy function $\alpha(s)$:
  - Continuous (when V is strictly concave)
  - Often monotone in state variables
- These properties guide the choice of numerical methods: smooth functions are easier to approximate.

Roadmap: From Theory to Computation

- We have established:
  - Equivalence between sequential and recursive formulations (Principle of Optimality)
  - Existence and uniqueness of solutions (Contraction Mapping Theorem)
  - Regularity of solutions (Theorem of the Maximum, concavity preservation)
- Now: **How do we compute V and α numerically?**
- Key challenge: V is a function, but computers work with finite objects.
- Trade-off: approximation accuracy ε vs. computational cost.
- Curse of dimensionality: cost grows exponentially in state dimension d.

A Numerical Dynamic Programming Algorithm

Numerical DP Algorithm

- We have reviewed some theoretical background on dynamic programming.
- Now, we will discuss its numerical implementation.
- Perhaps the most important solution algorithm to learn:
    - Wide applicability.
    - Many known results.
    - Template for other algorithms.
- Importance of keeping the ”curse of dimensionality” under control.

Value Function Iteration: Overview

- Well-known, basic algorithm of dynamic programming.
- We have tight convergence properties and bounds on errors.
- Well suited for parallelization.
- It will always (perhaps quite slowly) work.
- Key implementation questions:
    - How do we represent the value function numerically?
    - How do we construct the grid?
    - How do we interpolate between grid points?
    - How do we solve the maximization problem?

VFI: Algorithm Structure

- We begin with the Bellman operator:

$$B(V)(s) = \max_{a \in A(s)} \left[u(s,a) + \beta \int V(s') p(ds'|s,a) \right]$$

- Specify $V^{(0)}$ and apply Bellman operator:

$$V^{(1)}(s) = \max_{a \in A(s)} \left[u(s,a) + \beta \int V^{(0)}(s') p(ds'|s,a) \right]$$

- Iterate until convergence:

$$V^{(n+1)}(s) = B(V^{(n)})(s)$$
$$\|V^{(n+1)} - V^{(n)}\| < \text{tol}$$

- Convergence guaranteed by contraction mapping theorem.

VFI: From Continuous to Discrete

- Exact problem:

$$V(s) = \max_{a \in A(s)} \left[u(s,a) + \beta \int V(s') p(ds' | s, a) \right]$$

- Approximated problem:

$$\hat{V}(s_i) = \max_{a \in \hat{A}(s_i)} \left[u(s_i, a) + \beta \sum_{j=1}^{N} \hat{V}(s_j) \pi_{ij}(a) \right]$$

- Three sources of approximation error:
  - State space discretization: $s \to \{s_1, \ldots, s_N\}$
  - Transition probability approximation: $p(ds'|s,a) \to \pi_{ij}(a)$
  - Choice set restriction: $A(s) \to \hat{A}(s)$

VFI: Implementation Steps

1. **Setup:** Select accuracy level tol, initial guess $V_0^h \in \mathcal{W}^h$.
2. **Iteration:** At each grid point s_i, compute

$$V_{n+1}^h(s_i) = \max_{a \in \hat{A}(s_i)} \left[u(s_i, a) + \beta \sum_{j=1}^{N} V_n^h(s_j) \pi_{ij}(a) \right]$$

3. **Interpolation:** For off-grid evaluations of a, interpolate V_n^h.
4. **Convergence check:** If $\|V_{n+1}^h - V_n^h\| \leq$ tol, stop; else return to step 2.
5. **Policy extraction:** Recover optimal policy from final value function.

Function Space Approximation

- Basic idea: restrict to finite-dimensional function space that can be represented computationally.
- Define grid as finite collection of simplices $\{S^j\}$ such that:
  - $\cup S^j = S$ (covers state space)
  - $int(S^i) \cap int(S^j) = \emptyset$ for $i \neq j$ (non-overlapping interiors)
- Mesh size: $h = \max_j$ diam(S^j)
- Function space:

$$\mathcal{W}^h = \{V^h : S \to \mathbb{R} \mid V^h \text{ bounded, continuous, } DV^h \text{ constant in } int(S^j)\}$$

- Piecewise linear functions defined by values at vertices.

Grid Construction

Grid Construction: Why It Matters

- Grid quality directly affects:
    - Approximation accuracy
    - Computational cost
    - Convergence speed
- Key trade-off: more points $\Rightarrow$ higher accuracy but greater cost.
- Smart grid design: place points where they matter most.
- Curse of dimensionality: N points per dimension $\Rightarrow$ N^d total points.

Uniform Grids

- Simplest approach: equally spaced points.
- For interval $[a, b]$ with N points:

$$s_i = a + (i - 1)\frac{b - a}{N - 1}, \quad i = 1, \dots, N$$

- Advantages:
    - Simple to implement
    - Predictable structure
- Disadvantages:
    - Inefficient: same density everywhere
    - Poor for problems with high curvature in specific regions
    - Wastes points in flat regions of value function

Logarithmic Grids

- Concentrate points in regions of high curvature (typically near lower bound).
- For interval $[a, b]$ with N points:

$$s_i = a + (b-a)\frac{\exp(\lambda\frac{i-1}{N-1}) - 1}{\exp(\lambda) - 1}, \quad i = 1, \ldots, N$$

- Parameter $\lambda > 0$ controls curvature ($\lambda = 0$ gives uniform).
- Well-suited for capital grids in growth models:
    - Value function has high curvature near borrowing constraint
    - Flatter at high capital levels
- Common choice: $\lambda \in [2, 5]$ depending on model.

Chebyshev Nodes

- Optimal for polynomial interpolation.
- For interval $[-1, 1]$ with N points:

$$x_i = \cos\left(\frac{2i-1}{2N}\pi\right), \quad i = 1, \dots, N$$

- Transform to $[a, b]$: $s_i = \frac{a+b}{2} + \frac{b-a}{2} x_i$
- Key property: minimizes maximum interpolation error (minimax).
- Avoids Runge phenomenon (oscillations at boundaries).
- Particularly useful when combined with Chebyshev polynomial approximation.

Grid Construction: Comparison

	Uniform	Logarithmic	Chebyshev
Implementation	Simple	Moderate	Moderate
Point distribution	Even	Clustered at lower bound	Clustered at boundaries
Best for	Linear problems	Growth models	Polynomial approx.
Boundary behavior	Poor	Good at lower	Good at both

Practical advice:

- Start with logarithmic grid for consumption-saving problems.
- Use Chebyshev when employing spectral methods.
- Always test sensitivity to grid density.

Interpolation Schemes

Interpolation: Recap

- Covered in detail in "Numerical Methods for Macroeconomists."
- Why interpolation matters for VFI:
    - Choice variable a may not lie on grid
    - Need to evaluate $V(s')$ for off-grid states
    - Interpolation error propagates through iterations
- Key considerations:
    - Accuracy vs. computational cost
    - Preservation of economic properties (monotonicity, concavity)
    - Smoothness of resulting policy function

Piecewise Linear Interpolation

- Given grid $\{s_1, \dots, s_N\}$ and values $\{V_1, \dots, V_N\}$:

$$\hat{V}(s) = V_i + \frac{V_{i+1} - V_i}{s_{i+1} - s_i}(s - s_i), \quad s \in [s_i, s_{i+1}]$$

- Advantages:
    - Simple and fast
    - Preserves monotonicity automatically
    - Error bound: $O(h^2)$ where h is mesh size
- Disadvantages:
    - Not differentiable at grid points (kinks)
    - Can cause issues for optimization routines
    - Lower accuracy than higher-order methods

Cubic Spline Interpolation

- Piecewise cubic polynomials with continuous first and second derivatives.
- Achieves C^2 smoothness across entire domain.
- Advantages:
  - Higher accuracy: $O(h^4)$ error
  - Smooth derivatives aid optimization
  - Natural for economic problems (smooth preferences)
- Disadvantages:
  - May introduce spurious oscillations
  - Does not preserve monotonicity or concavity automatically
  - Higher computational cost
- Boundary conditions: natural spline ($V'' = 0$ at endpoints) most common.

Shape-Preserving Interpolation

- Maintains monotonicity and/or concavity of underlying function.
- Important for economic applications:
  - Value functions are typically concave
  - Policy functions are typically monotone
  - Violations can cause numerical instability
- Methods:
  - Monotone cubic Hermite (Fritsch-Carlson)
  - Schumaker quadratic spline (preserves both)
  - PCHIP (Piecewise Cubic Hermite Interpolating Polynomial)
- Trade-off: preserving shape may reduce approximation accuracy.

Interpolation: Comparison

	Linear	Cubic Spline	Shape-Preserving
Accuracy	$O(h^2)$	$O(h^4)$	$O(h^2)$ to $O(h^3)$
Smoothness	C^0	C^2	C^1
Monotonicity	Yes	No	Yes
Concavity	No	No	Possible
Speed	Fast	Moderate	Moderate

Recommendation:

- Start with linear for robustness and debugging.
- Switch to cubic spline for final results if no shape violations.
- Use shape-preserving if concavity/monotonicity violations occur.

Summary: VFI Implementation Checklist

1. **Grid construction:**
    - Choose grid type (uniform, logarithmic, Chebyshev)
    - Select number of points (start coarse, refine)
    - Discretize exogenous shocks (Tauchen/Rouwenhorst)
2. **Interpolation:**
    - Start with linear for debugging
    - Upgrade to cubic spline or shape-preserving
3. **Optimization:**
    - Golden section search (univariate)
    - Exploit concavity when applicable
4. **Convergence:**
    - Monitor sup norm of value function changes
    - Check policy function stability
    - Verify error bounds with grid refinement

Solution Methods Based on the Euler Equations

From VFI to Euler Equation Methods

- VFI iterates on the value function $V(s)$.
- Alternative: iterate directly on the policy function $g(s)$.
- Advantages of policy-based methods:
    - Policy function often simpler (lower curvature) than value function.
    - Avoid maximization step at each iteration.
    - Can be faster for certain problem classes.
- Key tool: Euler equations from first-order conditions.

Deriving Euler Equations: Setup

- Consider the one-sector growth model:

$$\max_{\{c_t, k_{t+1}\}} \sum_{t=0}^{\infty} \beta^t u(c_t)$$

$$\text{s.t. } c_t + k_{t+1} = A k_t^{\alpha} + (1-\delta) k_t$$

- Lagrangian approach:

$$\mathcal{L} = \sum_{t=0}^{\infty} \beta^t \left[u(c_t) + \lambda_t \left(A k_t^{\alpha} + (1-\delta) k_t - c_t - k_{t+1} \right) \right]$$

- First-order conditions yield the Euler equation.

Deriving Euler Equations: FOCs

- FOC with respect to c_t:

$$\frac{\partial \mathcal{L}}{\partial c_t} = \beta^t u'(c_t) - \beta^t \lambda_t = 0 \quad \Rightarrow \quad u'(c_t) = \lambda_t$$

- FOC with respect to k_{t+1}:

$$\frac{\partial \mathcal{L}}{\partial k_{t+1}} = -\beta^t \lambda_t + \beta^{t+1} \lambda_{t+1} \left(\alpha A k_{t+1}^{\alpha-1} + 1 - \delta \right) = 0$$

- Combining:

$$u'(c_t) = \beta u'(c_{t+1}) \left(\alpha A k_{t+1}^{\alpha-1} + 1 - \delta \right)$$

- This is the **intertemporal Euler equation**.

Euler Equation: Interpretation

- With $u(c) = \log(c)$, the Euler equation becomes:

$$\frac{1}{c_t} = \beta \frac{\alpha A k_{t+1}^{\alpha-1} + 1 - \delta}{c_{t+1}}$$

- Using the budget constraint $c_t = Ak_t^\alpha + (1-\delta)k_t - k_{t+1}$:

$$\frac{1}{Ak_t^\alpha + (1-\delta)k_t - k_{t+1}} = \beta \frac{\alpha A k_{t+1}^{\alpha-1} + 1 - \delta}{Ak_{t+1}^\alpha + (1-\delta)k_{t+1} - k_{t+2}}$$

- Interpretation: marginal cost of saving today equals discounted marginal benefit tomorrow.

Policy Function Representation

- By the principle of optimality, there exists a unique $g(k)$ such that:

$$k_{t+1} = g(k_t)$$

- The sequence $\{k_t\}$ generated recursively from k_0 solves the planner's problem.
- Substituting into the Euler equation:

$$\frac{1}{Ak^{\alpha} + (1-\delta)k - g(k)} = \beta \frac{\alpha A g(k)^{\alpha-1} + 1 - \delta}{Ag(k)^{\alpha} + (1-\delta)g(k) - g(g(k))}$$

- This is a **functional equation** in $g(\cdot)$.

Time Iteration: Basic Idea

- Define the Euler equation operator. Given policy g_n, find g_{n+1} such that:

$$u'(c(k;g_{n+1})) = \beta u'(c(g_{n+1}(k);g_n)) \cdot R(g_{n+1}(k))$$

  where $R(k') = \alpha A(k')^{\alpha-1} + 1 - \delta$ is the gross return.
- Interpretation:
  - g_n: “old” policy function (tomorrow’s behavior)
  - g_{n+1}: “new” policy function (today’s optimal response)
- Iterate until $\|g_{n+1} - g_n\| <$ tol.

Time Iteration: Algorithm

1. **Initialize:** Choose grid $\{k_1, \dots, k_N\}$, initial guess $g_0(k)$.
2. **At each grid point k_i:** Solve for k' satisfying

$$u'\left(Ak_i^\alpha + (1-\delta)k_i - k'\right) = \beta u'\left(Ak'^\alpha + (1-\delta)k' - g_n(k')\right) R(k')$$

3. **Update:** Set $g_{n+1}(k_i) = k'$ for all i.
4. **Interpolate:** Construct $g_{n+1}(\cdot)$ over full domain.
5. **Check convergence:** If $\|g_{n+1} - g_n\|_\infty <$ tol, stop; else return to step 2.

Time Iteration: Implementation Details

- At each grid point, solve a **nonlinear equation** in k':

$$F(k') = u'(c) - \beta u'(c')R(k') = 0$$

- Root-finding methods:
    - Bisection: robust but slow
    - Newton-Raphson: fast but requires derivative
    - Brent's method: good compromise
- Key difference from VFI: root-finding replaces maximization.
- Computational cost per iteration often lower than VFI.

Time Iteration with Stochastic Shocks

- With productivity shock z, Euler equation becomes:

$$u'(c(k,z)) = \beta \mathbb{E}\left[u'(c(k',z'))R(k',z') \mid z\right]$$

- Time iteration: given $g_n(k,z)$, solve for $g_{n+1}(k,z)$:

$$u'(c) = \beta \sum_{z'} \pi(z'|z) u'(c(g_{n+1}(k,z),z';g_n))R(g_{n+1}(k,z),z')$$

- At each (k_i, z_j): solve one nonlinear equation.
- Expectation computed using discretized transition matrix $\pi(z'|z)$.

Convergence: The Key Question

- Does time iteration converge to a fixed point?
- VFI: convergence guaranteed by contraction mapping theorem.
- Time iteration: Bellman operator structure not directly available.
- Two approaches to establish convergence:
    - Show contraction property (Coleman 1990)
    - Use monotonicity properties (Greenwood-Huffman 1995)
- Greenwood-Huffman approach particularly useful for models with externalities where contraction is hard to verify.

Greenwood-Huffman: Motivation

- Consider economies with externalities:

$$y_t = F(k_t, K_t, z_t)$$

- Individual capital k_t vs. aggregate capital K_t (taken as given).
- Individual Euler equation:

$$u'(c_t) = \beta E_t \left[u'(c_{t+1}) F_1(k_{t+1}, K_{t+1}, z_{t+1})\right]$$

- In equilibrium: $k_t = K_t$.
- Challenge: prove existence of recursive equilibrium functions g, G.

Greenwood-Huffman: The Operator

- Define operator T on policy functions.
- Let $H^0 = 0$. Define $H^{j+1}(K, z) = x$ that solves:

$$u'(F(K,K,z) - x) = \beta E\left[u'(F(x,x,z') - H^j(x,z'))F_1(x,x,z')\right]$$

- This generates a sequence of functions $\{H^j\}$.
- Key question: Does this sequence converge to a limit H^*?
- If yes, H^* characterizes the recursive equilibrium.

Greenwood-Huffman: Unique Solution for x

- At each iteration, need unique solution x to:

$$u'(F(K,K,z)-x)=\beta E\left[u'(F(x,x,z')-H^j(x,z'))F_1(x,x,z')\right]$$

- Left-hand side: **increasing** in x (since $u''<0$).
- Right-hand side: **decreasing** in x under appropriate conditions.
  - Requires restrictions on cross-derivatives of F.
  - Slope: $u''(F_1+F_2-\frac{\partial H}{\partial x})F_1+u'(F_{11}+F_{12})$
- Increasing meets decreasing $\Rightarrow$ unique intersection.

Greenwood-Huffman: Monotonicity Approach

- Key insight: avoid contraction mapping, use **monotonicity**.
- Two ways to find fixed points of operators:
  - Contraction properties (standard DP approach)
  - Monotonicity properties (Greenwood-Huffman approach)
- If for each (K, z) the sequence $\{H^j(K, z)\}_j$ is:
  - Increasing (or decreasing)
  - Bounded
- Then a limit exists by monotone convergence.

Greenwood-Huffman: Monotonicity of Operator

- Recall: $H^{j+1} = TH^j$ solves:

$$u'(F(K,K,z) - x) = \beta E\left[u'(F(x,x,z') - H^j(x,z'))F_1(x,x,z')\right]$$

- LHS increasing in x, RHS decreasing in x.
- Claim: If $H^j > G^j$ pointwise, then $TH^j > TG^j$.
- Proof intuition:
  - Higher $H^j \Rightarrow$ lower RHS for given x
  - To restore equality, need higher x
  - Hence $TH^j > TG^j$
- Operator T is **monotone**.

Greenwood-Huffman: Convergence Result

- Starting from $H^0 = 0$:
  - Sequence $\{H^j(K, z)\}_j$ is increasing
  - Bounded above (by resource constraints)
- By monotone convergence: $H^j \to H^*$.
- Additional conditions ensure bounds on derivatives $\{\frac{\partial H^j}{\partial K}\}_j$.
- Sequence is equicontinuous $\Rightarrow$ limit H^* is continuous.
- H^* satisfies the equilibrium Euler equation by construction.

Greenwood-Huffman: Equilibrium Characterization

- The limit function H^* solves:

$$u'(c(K,K,x,z)) = \beta E\left[u'(c(x,x,H^*(x,z'),z'))F_1(x,x,z')\right]$$

- Given this aggregate law of motion, individual problem has unique solution $k' = q(k,K,z)$.
- When $k = K$: individual policy $q = H^*$ solves individual Euler equation.
- Therefore H^* characterizes the recursive competitive equilibrium.

Implications for Time Iteration

- Greenwood-Huffman provides theoretical foundation for Euler-based iteration.
- Convergence via monotonicity:
  - Does not require contraction property
  - Works for models with externalities
  - Requires monotone operator and bounded sequences
- Practical implications:
  - Start from $g_0 = 0$ (or steady state)
  - Monitor monotonicity of sequence
  - Convergence guaranteed under G-H conditions
- For standard growth models without externalities: Coleman (1990) contraction results also apply.

Convergence Properties: Summary

- **Contraction approach** (Coleman 1990):
  - Applies to standard neoclassical models
  - Geometric convergence rate
  - Requires verifying contraction conditions
- **Monotonicity approach** (Greenwood-Huffman 1995):
  - Applies to models with externalities
  - Convergence via bounded monotone sequences
  - Requires monotone operator and appropriate bounds
- Both approaches: time iteration converges to correct equilibrium under appropriate conditions.

Euler Equation Residuals

- Suppose we have an approximate policy $\tilde{g}(k)$ satisfying:

$$\frac{1}{Ak^\alpha + (1-\delta)k - \tilde{g}(k)} \approx \beta \frac{\alpha A(\tilde{g}(k))^{\alpha-1} + 1 - \delta}{A(\tilde{g}(k))^\alpha + (1-\delta)\tilde{g}(k) - \tilde{g}(\tilde{g}(k))}$$

- Define the **Euler equation residual**:

$$\mathcal{E}(k) = 1 - \frac{\beta u'(c')R(k')}{u'(c)}$$

- Question: Does small $\mathcal{E}(k)$ imply good approximation?
- Answer: Yes, under regularity conditions (Santos, Econometrica 2000).

Euler Equation Errors: Formal Definition

- For general problems, define residual using envelope conditions:

$$\mathcal{R}(k) = v_2(k, \hat{g}(k)) + \beta v_1(\hat{g}(k), \hat{g}(\hat{g}(k)))$$

  where v_1, v_2 are partial derivatives of the return function.
- Goal: find policy $\hat{g}$ achieving:

$$\max_k |\mathcal{R}(k)| \leq \epsilon$$

- To make computable: parameterize policy function by finite vector $\zeta \in \mathbb{R}^n$.
- Choose ζ to minimize residuals (projection methods, Topic 9).

Unit-Free Euler Equation Errors

- Raw residuals depend on units; hard to interpret.
- **Unit-free error** (Judd 1992):
$$\mathsf{EE}(k) = \left|1 - \frac{c^*}{c}\right|$$
where c^* solves Euler equation exactly given $\hat{g}$, and $c = c(k; \hat{g})$.
- Interpretation: fraction of consumption household would change if re-optimizing.
- Typical targets:
  - $\max |\mathsf{EE}(k)| < 10^{-3}$: acceptable
  - $\max |\mathsf{EE}(k)| < 10^{-5}$: good
  - $\max |\mathsf{EE}(k)| < 10^{-7}$: excellent

Comparison: VFI vs. Time Iteration

	VFI	Time Iteration
Iterates on	Value function	Policy function
Per-iteration step	Maximization	Root-finding
Convergence proof	Contraction	Contraction or Monotonicity
Convergence speed	Slower	Often faster
Accuracy control	Grid density	Euler residuals
Externalities	Complex	Natural (G-H)

When to Use Each Method

- **Use VFI when:**
  - Robustness is priority (always converges to optimum)
  - Problem has non-differentiabilities
  - Occasionally binding constraints present
  - Need value function for welfare analysis
- **Use Time Iteration when:**
  - Speed is priority
  - Problem is smooth and well-behaved
  - Only policy function needed for simulation
  - Models with externalities (G-H framework)
- Best practice: implement both, cross-validate results.

Summary: Euler Equation Methods

- Euler equations derived from FOCs characterize optimal policy.
- Time iteration: fixed-point iteration on policy function space.
- Convergence foundations:
  - Coleman (1990): contraction for standard models
  - Greenwood-Huffman (1995): monotonicity for models with externalities
- Euler equation residuals:
  - Measure approximation quality
  - Unit-free errors interpretable as consumption equivalents
  - Small residuals $\Rightarrow$ small policy errors (Santos 2000)
- Next: Endogenous Grid Method (EGM) avoids root-finding entirely.