Lecture 1: Introduction and Dynamic Programming

Fatih Guvenen
January 2026

Four Components of a Quantitative Project

1. Model specification:
   - Preferences, technology, demographic structure, equilibrium concept, frictions, driving forces, etc.
2. Numerical solution:
   - Programming language, algorithms, accuracy vs speed, etc.
3. Calibration/Estimation:
   - Simulation-based estimation, global optimization
4. Analyzing the solved model:
   - Policy experiments/counterfactuals, welfare analysis, transitions, etc.

Prototypical Problem You Will Need to Solve

1. A **Dynamic Programming** problem, with:
   - 2 choice variables, 2-4 continuous state variables
   - 1-2 discrete state variables
   - Fixed costs, adjustment costs, irreversibilities, etc.

A Word about Programming Languages

- Choice of programming language is critical for successfully solving problems like the one above.
- Three (broad) types of programming languages
  - Low-level/Compiled languages: Fortran, C/C++
  - High level/Interpreted languages: Matlab, Python, R, Stata, etc.
  - High-level language with option to compile: Julia.

Comparison Beyond Speed

- Comparison for large-scale problems (i.e., the prototypical problem above):

	Compiled	Interpreted
Speed	10 to 100 times faster	Much slower

Dynamic Programming: Introduction

GOAL: Solve the Bellman Equation

$$V(k,z) = \max_{c,k'} [u(c) + \beta \mathbb{E}(V(k',z')|z)]$$

$$c + k' = (1+r)k + z$$

$$z' = \rho z + \eta \qquad \eta \overset{i.i.d.}{\sim} F(.)$$

- Solution involves finding unknown functions: $c(k,z), k'(k,z), V(k,z)$

Contraction Mapping Theorem

- **Definition (Contraction Mapping)** Let (S, d) be a metric space and $T : S \to S$ be a mapping of S into itself. T is a contraction mapping with modulus β, if for some $\beta \in (0, 1)$ we have

$$d(Tv_1, Tv_2) \leq \beta d(v_1, v_2)$$

for all $v_1, v_2 \in S$.

Qualitative Properties of v^*

- We cannot apply CMT in certain cases, because the particular set we are interested in is not a complete metric space.

A Prototype Problem

$$V(k,z) = \max_{c,k'} \left[u(c) + \beta \int V(k',z') f(z'|z) dz' \right]$$
$$c + k' = (1+r)k + z$$
$$z' = \rho z + \eta$$

- CMT tells us to start with an *appropriate guess* V_0, then repeatedly solve the problem on the RHS.

Let's Start with a Simple Analytical Example

Neoclassical Growth Model

- Consider the special case with log utility, Cobb-Douglas production and full depreciation:

$$\begin{aligned} V(k) &= \max_{c,k'} \{\log c + \beta V(k')\} \\ \text{s.t} \quad c &= Ak^{\alpha} - k' \end{aligned}$$

I. Backward Induction (Brute Force)

- If $t = T < \infty$, in the last period we would have: $V_0(k) \equiv 0$ for all k. Therefore:

$$V_1(k) = \max_{k'} \left\{ \log(Ak^\alpha - k') + \underbrace{\beta V_0(k')}_{\equiv 0} \right\}$$

II. Guess and Verify (Value Function)

- But there is a more direct approach.

II. Guess and Verify (Value Function)

- Let $LHS = a + b \log k$. Plug in the expression for k' into the RHS:

$$\begin{aligned} RHS &= \log\left(Ak^{\alpha} - \frac{\beta b}{1+\beta b} Ak^{\alpha}\right) + \beta\left(a + b\log\left(\frac{\beta b}{1+\beta b} Ak^{\alpha}\right)\right) \\ &= \underbrace{(1+\beta b)\log A + \log\left(\frac{1}{1+\beta b}\right) + a\beta + b\beta \log\left(\frac{\beta b}{1+\beta b}\right)}_{\text{CONSTANT!}} \\ &\quad + \alpha(1+\beta b) \times \log k \end{aligned}$$

Guess and Verify as a Numerical Tool

- Although this was a very special example, the same general idea underlies many numerical methods:

III. Guess and Verify (Policy Functions)

- Let the policy rule for savings be: $k' = g(k)$. The Euler equation is:

$$\frac{1}{Ak^{\alpha} - g\left(k\right)} - \frac{\beta\alpha A\left(g\left(k\right)^{\alpha-1}\right)}{A\left(g\left(k\right)^{\alpha} - g\left(g\left(k\right)\right)\right)} = 0 \quad \text{for all } k.$$

  which is a functional equation in $g\left(k\right)$.

Standard VFI

- Standard Value Function Iteration is simply the application of the Contraction Mapping Theorem

Algorithmus 1 : Standard Value Function Iteration

1. Set $n = 0$. Choose an initial guess $V_0 \in S$.
2. Obtain V_{n+1} by applying the mapping: $V_{n+1} = TV_n$, which entails (i) maximizing the right-hand side of the Bellman equation and (ii) then plugging in the decision rule obtained into the RHS.
3. Stop if convergence criteria satisfied: $|V_{n+1} - V_n| < \texttt{toler}$. Otherwise, increase n and return to step 2.

- We will call (i) the maximization step, and (ii) the evaluation step.

Quick Digression: How To Represent $V(k, z)$ on a Computer?

- A value function with continuous state variables—e.g., $V(k, z)$—is an infinite-dimensional object.

Quick Digression: How To Represent $V(k, z)$ on a Computer?

- Another approach would be to use parametric families of analytical functions:
  - Polynomials (Chebyshev, Hermite, Legendre, etc.), or combinations of power functions, logs, exponentials, etc.

Now: Apply VFI to Neoclassical Growth Model

Consider the neoclassical growth model:

$$V(k,z) = \max_{c,k'} \left\{ \frac{c^{1-\gamma}}{1-\gamma} + \beta \mathbb{E}\left(V(k',z') \,|z\right) \right\}$$

$$\text{s.t} \quad c + k' = e^{z} k^{\alpha} + (1-\delta)k \tag{P1}$$

$$z' = \rho z + \eta', \qquad k' \geq \underline{k}.$$

VFI is (Very!) Slow. How to Speed It Up?

- Recall that for a contraction mapping: $d(Tv_1, Tv_2) \leq \beta d(v_1, v_2)$.
- So when $\beta \approx 1$ (say $\beta = 0.99$ or 0.999), VFI can be **very slow**.
- Three methods to accelerate:

Howard's Policy Iteration

- Howard's Policy iteration follows from **two key observations** about VFI:
  - The maximization step (eq. 1) is **typically much more costly** (in computational time) than the evaluation step (eq. 2).
  - But the latter uses the updated decision rule, $\tilde{s}_n(k_i, z_j)$, **for only one period** (since savings decisions after tomorrow is embedded in V_n on the RHS).
- **Policy Iteration**: Repeat the evaluation step multiple times between each maximization step.
- **Definition:** For a given value function J and a decision rule w, define **Howard's mapping** $\widetilde{T}$ as the operator that "plugs in w_n" into the RHS of the Bellman equation:

$$\widetilde{T}_w J(k_i, z_j) = \frac{\left(e^{z_j} k_i^{\alpha} + (1-\delta)k_i - w(k_i, z_j)\right)^{1-\gamma}}{1-\gamma} + \beta \mathbb{E}\left(J\left(w(k_i, z_j), z'\right) | z_j\right) \tag{3}$$

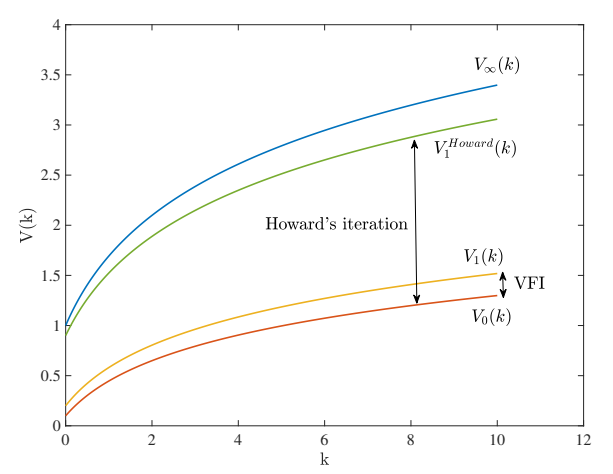
Howard's Policy Iteration: Cont'd

- We will be interested in applying the Howard mapping repeatedly, so for $m = 1, \ldots, M$, let

$$J_{m+1} \equiv \widetilde{T}_w J_m(k_i, z_j) \tag{4}$$

denote the updated value function.

VFI vs Howard's Algorithm



Two Properties of Howard's Algorithm

Puterman and Brumelle (1979) show that:

- Policy iteration is equivalent to the Newton-Kantarovich method applied to dynamic programming.

Drawbacks of Howard's Policy Iteration

1. **Quadratic convergence** is a bit misleading: this is the rate in n (number of maximization steps)

VFI with Modified Policy Iteration (MPI) Algorithm

- Modify *Step 3* of Howard's algorithm above:
  - **Modified Howard Step**: Set $J_0 \equiv V_n$, and iterate on $J_{m+1} \equiv \widetilde{T}_{\tilde{s}_n} J_m(k_i, z_j)$ for $m = 0, 1, 2, \ldots, M < \infty$. Choose a moderate value for M (by experimentation and smaller for more challenging problems). Then, set $V_{n+1} = J_M$.

Dampened VFI Algorithm

Modify Step 2 of the VFI algorithm as follows:

2*. Obtain J_{n+1} from V_n by applying the standard *Bellman mapping*:

$$J_{n+1} = TV_n,$$

(i.e., maximize RHS of the Bellman equation and evaluate with the new optimal policy.)

Error Bounds: Background

- In iterative numerical algorithms, we need a stopping rule.
- In dynamic programming, we want to know how far we are from the true solution in each iteration.

Two Remarks

1. The CMT bound is for the worst case scenario (sup-norm). If V^* varies over a wide range, this bound will (typically) be misleading—too pessimistic.
   - Consider $u(c) = \frac{c^{1-\alpha}}{1-\alpha}$ with $\alpha = RRA = 10$. V will cover an enormous range of values. Bound will be too pessimistic.

MacQueen-Porteus Bounds

Consider this alternative formulation of a dynamic programming problem:

$$V(x_i) = \max_{y \in \Gamma(x_i)} \left[U(x_i, y) + \beta \sum_{j=1}^{J} \pi_{ij}(y) V(x_j) \right], \tag{5}$$

- State space is discrete.
- But choices are continuous.
- Allows for simple modeling of interesting problems.
- Popular formulation in other fields using dynamic programming.
  - See, e.g., Bertsekas and Shreve (1978) which is a wonderful book on DP, or Bertsekas and Ozdaglar (2009) for a more up to date comprehensive treatment.

MacQueen-Porteus Bounds

Theorem 1
[MacQueen-Porteus bounds] *Consider*

$$V(x_i) = \max_{y \in \Gamma(x_i)} \left[U(x_i, y) + \beta \sum_{j=1}^{J} \pi_{ij}(y) V(x_j) \right], \tag{6}$$

define

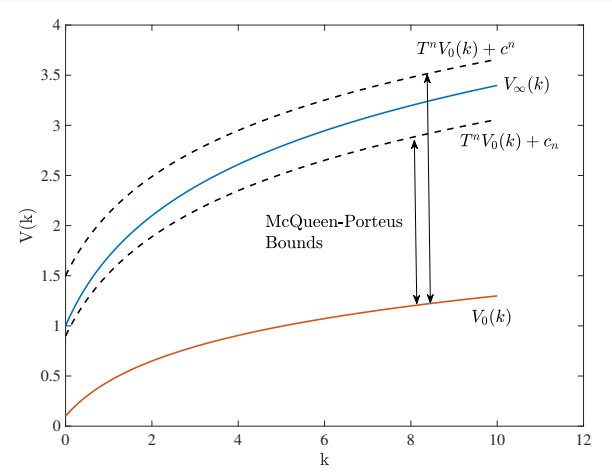
$$\underline{c}_n \equiv \frac{\beta}{1-\beta} \times \min\left[V_n - V_{n-1}\right] \qquad \overline{c}_n \equiv \frac{\beta}{1-\beta} \times \max\left[V_n - V_{n-1}\right] \tag{7}$$

Then, for all $\overline{x} \in X$, *we have:*

$$T^n V_0(\overline{x}) + \underline{c}_n \leq V^*(\overline{x}) \leq T^n V_0(\overline{x}) + \overline{c}_n. \tag{8}$$

Furthermore, *with each iteration, the two bounds approach the true solution* **monotonically.**

VFI versus McQueen-Porteus Bounds



MQP Bounds: Comments

- MQP bounds can be quite tight.
- Example: Suppose $V_n(\overline{x}) - V_{n-1}(\overline{x}) = \alpha$ for all $\overline{x}$ and that $\alpha = 100$ (a large number).

VFI Stopping Rule with MQP Bounds

Algorithmus 3 : VFI Stopping Rule with MQP Error Bounds

[*Step 3′:*] Stop when $\overline{c}_n - \underline{c}_n < \texttt{toler}$. Then take the final estimate of V^* to be either the median

$$\tilde{V} = T^n V_0 + \left(\frac{\overline{c}_n + \underline{c}_n}{2} \right)$$

or the mean (i.e., average error bound across states):

$$\hat{V} = T^n V_0 + \frac{\beta}{n(1-\beta)} \sum_{i=1}^{n} \left(T^n V_0(\overline{x}_i) - T^{n-1} V_0(\overline{x}_i) \right).$$

VFI Acceleration with MQP Bounds

Algorithmus 4 : VFI Acceleration with MQP Error Bounds

[*Step 2'*:] After every m iteration (for e.g., $m = 1, 2, ..., M$) on the VFI algorithm, instead of the usual VFI updating $V_{n+1} = TV_n$, take one "big" MQP step by setting:

$$V_{n+1} = TV_n + \left(\frac{\overline{c}_n + \underline{c}_n}{2} \right)$$

[*Step 3'*:] Same as above (in VFI Stopping Rule with MQP Bound algorithm)

MQP: Convergence Rate

- Bertsekas (1987) derives the convergence rate of MQP bounds algorithm
- It is proportional to the **subdominant eigenvalue** of $\pi_{ij}(y^*)$ (the transition matrix evaluated at optimal policy).

Benchmarking MQP and MPI: Parameters

- We will consider:
  - $\beta = 0.95, 0.99, 0.999$
  - $RRA = 1, 5$
  - MPI: $m = 0, 50, 500$
  - MPQ: $m = 1$. (MQP update step in every VFI iteration).
  - $V_0 = 0$ (inefficient choice)

Benchmarking MQP and MPI

β : time discount factor, m : # of Howard iterations, γ : relative risk aversion.

Table 1: Mc-Queen Porteus Bounds and Policy Iteration

$\beta \rightarrow$	0.95			0.99			0.999		
m :	0	50	500	0	50	500	0	50	500
MQP	(RRA) $\gamma = 1$								
No	14.99	1.07	1.00*	26.48	1.28	1.00*	33.28	1.41	1.00*
Yes	0.32	0.60	0.79	0.10	0.23	0.27	0.01	0.03	0.04
	(RRA) $\gamma = 5$								
No	13.03	0.96	1.00*	26.77	1.28	1.00*	33.37	1.43	1.00*
Yes	0.67	0.67	0.69	0.14	0.24	0.30	0.02	0.04	0.05

*Time normalized to 1 for the Howard run with $m = 500$ and without MQP.

Takeaways from the Example

1. Relative to plain VFI ($m = 0$):
   - **MPI** alone speeds up by 13 to 33 times

Takeaways (Cont'd)

- **Important note:** These numbers are not written in stone! Your mileage will vary depending on the complexity of the problem and other factors.

Endogenous Grid Method

- In standard VFI, we have FOC:

$$c^{-\gamma} = \beta \mathbb{E}\left(V_k\left(k', z'\right) | z_j\right).$$

- **View the problem differently:**

$$V(k, z_j) = \max_{c,k'} \left\{ \frac{c^{1-\gamma}}{1-\gamma} + \beta \mathbb{E}\left(V(k_i', z') | z_j\right) \right\}$$
$$\text{s.t} \quad c + k_i' = z_j k^{\alpha} + (1-\delta)k \tag{P3}$$
$$\ln z' = \rho \ln z_j + \eta',$$

- **Trick 2:** Define

$$Y \equiv zk^{\alpha} + (1-\delta)k \tag{11}$$

and rewrite the Bellman equation (without discretization) as:

$$\mathcal{V}(Y,z) = \max_{k'} \left\{ \frac{(Y-k')^{1-\gamma}}{1-\gamma} + \beta \mathbb{E}\left(\mathcal{V}(Y',z') \,|z\right) \right\}$$

$$\textbf{s.t} \qquad \ln z' = \rho \ln z + \eta'.$$

EGM

- Plug $\mathbb{V}$ back into modified Bellman Equation:

$$\mathcal{V}(Y, z) = \max_{k'} \left\{ \frac{(Y - k')^{1-\gamma}}{1-\gamma} + \mathbb{V}(k'_i, z_j) \right\}$$

EGM: The Algorithm

0: Set $n = 0$. Construct a grid for tomorrow's capital and today's shock: (k'_i, z_j). Choose an initial guess $\mathbb{V}^0(k'_i, z_j)$.

EGM: The Algorithm (Cont'd)

3: Interpolate $\mathcal{V}^{n+1}$ to obtain its values on a grid of tomorrow's end-of-period resources: $Y' = z'(k_i')^{\alpha} + (1-\delta)k_i'$.

Comments

- Whenever EGM can be applied, it should be your default choice. It can easily be 1-2 orders of magnitude faster than VFI with acceleration methods.
- Extensions and Limitations:
  - Two choice variables can be handled with some loss of efficiency. See Barillas and Fernandez-Villaverde (JEDC 2007) and Maliar and Maliar (2013).
  - Two state variables: currently no "simple" solution that keeps accuracy intact.
  - Borrowing constraints: Very easy to deal with.

Is This Worth the Trouble? Yes!

	β			
Utility	0.95	0.98	0.99	0.995
VFI	28.9	74	119	247
VFI + Howard	7.17	18.2	29.5	53
VFI + Howard + MQP	7.17	16.5	26	38
VFI + Howard + MQP +100 grid	2.15	5.2	8.2	12
EGM (expanding grid curv=2)	0.38	0.94	1.92	4

Table 2: Time for convergence (seconds)

- RRA=2; 300 points in capital grid, expanding grid with exponent of 3.