Lecture 1: Introduction and Dynamic Programming

Fatih Guvenen
University of Minnesota

November 2023

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.

This Class: 2 & 3

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 Probably 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.

Prototypical Problem You Will Probably 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.
   - Which will be embedded in...

Prototypical Problem You Will Probably 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.
   - Which will be embedded in...
2. A **GE model**, possibly with aggregate shocks, and
   - two or more equilibrium pricing functions to solve as a function of aggregate state and wealth distribution
   - endogenous laws of motion to solve for
   - stationary distributions to find

Prototypical Problem You Will Probably 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.
   - Which will be embedded in...
2. A **GE model**, possibly with aggregate shocks, and
   - two or more equilibrium pricing functions to solve as a function of aggregate state and wealth distribution
   - endogenous laws of motion to solve for
   - stationary distributions to find
   - Which will be embedded in...

Prototypical Problem You Will Probably 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.
   - Which will be embedded in...
2. A **GE model**, possibly with aggregate shocks, and
   - two or more equilibrium pricing functions to solve as a function of aggregate state and wealth distribution
   - endogenous laws of motion to solve for
   - stationary distributions to find
   - Which will be embedded in...
3. An **estimation/calibration problem** with 5 to 15 parameters by matching moments
   - where moments can have kinks or jumps in parameters
   - the objective is likely to have multiple local minima (sometimes hundreds of them)

A Word about Programming Languages

- Choice of programming language is critical for successfully solving a problem 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.

A Word about Programming Languages

- Choice of programming language is critical for successfully solving a problem 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.
- One important difference: **Speed!**

A Word about Programming Languages

- Choice of programming language is critical for successfully solving a problem 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.
- One important difference: **Speed!**
- In scientific disciplines where computational demands are high, compiled languages are much more popular.
- Julia is a great option: A more modern language that can be fast if you know how to optimize it. But it requires work & experience to make use of its speed. (Still not as fast as C/Fortran though)

Comparison Beyond Speed

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

	Compiled	Interpreted
Speed	10 to 100 times faster	Much slower

Comparison Beyond Speed

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

	Compiled	Interpreted
Speed	10 to 100 times faster	Much slower
Ease of coding	Higher set up cost	Lower set up cost
	But often clearer code	Usually simpler syntax

Comparison Beyond Speed

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

	Compiled	Interpreted
Speed	10 to 100 times faster	Much slower
Ease of coding	Higher set up cost	Lower set up cost
	But often clearer code	Usually simpler syntax
Ease of debug. complex code	Compiler catches bugs	Errors harder to find

Comparison Beyond Speed

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

	Compiled	Interpreted
Speed	10 to 100 times faster	Much slower
Ease of coding	Higher set up cost	Lower set up cost
	But often clearer code	Usually simpler syntax
Ease of debug. complex code	Compiler catches bugs	Errors harder to find
Control over memory, CPU	More customizable/scalable	Less control

Comparison Beyond Speed

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

	Compiled	Interpreted
Speed	10 to 100 times faster	Much slower
Ease of coding	Higher set up cost	Lower set up cost
	But often clearer code	Usually simpler syntax
Ease of debug. complex code	Compiler catches bugs	Errors harder to find
Control over memory, CPU	More customizable/scalable	Less control
Availability of scientific libraries	Very large & often free	Large but can require fee

Comparison Beyond Speed

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

	Compiled	Interpreted
Speed	10 to 100 times faster	Much slower
Ease of coding	Higher set up cost	Lower set up cost
	But often clearer code	Usually simpler syntax
Ease of debug. complex code	Compiler catches bugs	Errors harder to find
Control over memory, CPU	More customizable/scalable	Less control
Availability of scientific libraries	Very large & often free	Large but can require fee

- **Important note**: Linux/Mac are much more efficient at memory management than Windows. So, for large problems with *very* large data objects (like large matrices or arrays), your code can run much faster using the former.

Dynamic Programming: Goal

Solve:

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

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

$$z' = \rho z + \eta$$

- Questions:

Dynamic Programming: Goal

Solve:

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

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

$$z' = \rho z + \eta$$

- Questions:
  1. Does a solution exist?

Dynamic Programming: Goal

Solve:

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

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

$$z' = \rho z + \eta$$

- Questions:
  1. Does a solution exist?
  2. Is it unique?

Dynamic Programming: Goal

Solve:

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

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

$$z' = \rho z + \eta$$

- Questions:
  1. Does a solution exist?
  2. Is it unique?
  3. If the answers to (1) and (2) are yes: how do we find this solution?

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 $v_1, v_2 \in S$.

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 $v_1, v_2 \in S$.

- **Contraction Mapping Theorem:** Let (S, d) be a complete metric space and suppose that $T : S \to S$ is a contraction mapping. Then, T has a unique fixed point $v^* \in S$ such that

$$Tv^* = v^* = \lim_{N \to \infty} T^N v_0$$

  for all $v_0 \in S$.

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 $v_1, v_2 \in S$.

- **Contraction Mapping Theorem:** Let (S, d) be a complete metric space and suppose that $T : S \to S$ is a contraction mapping. Then, T has a unique fixed point $v^* \in S$ such that

$$Tv^* = v^* = \lim_{N \to \infty} T^N v_0$$

  for all $v_0 \in S$.

- The beauty of CMT is that it is a *constructive* theorem: it not only tells us the existence/uniqueness of v^* but it also shows us how to find it!

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.

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.
- The following corollary comes in handy in those cases.

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.
- The following corollary comes in handy in those cases.
- **Corollary:** Let (S, d) be a complete metric space and $T : S \longrightarrow S$ be a contraction mapping with $Tv^* = v^*$.
  a. If $\overline{S}$ is a closed subset of S, and $T(\overline{S}) \subset \overline{S}$, then $v^* \in \overline{S}$.
  b. If, in addition, $T(\overline{S}) \subset \overline{\overline{S}} \subset \overline{S}$, then $v^* \in \overline{\overline{S}}$.

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.
- The following corollary comes in handy in those cases.
- **Corollary:** Let (S, d) be a complete metric space and $T : S \longrightarrow S$ be a contraction mapping with $Tv^* = v^*$.
  a. If $\overline{S}$ is a closed subset of S, and $T(\overline{S}) \subset \overline{S}$, then $v^* \in \overline{S}$.
  b. If, in addition, $T(\overline{S}) \subset \overline{\overline{S}} \subset \overline{S}$, then $v^* \in \overline{\overline{S}}$.
- $\overline{\overline{S}}$ = {continuous, bounded, strictly concave}. Not a complete metric space. $\overline{S}$ = {continuous, bounded, weakly concave} is.
  - So we need to be able to establish that T maps elements of $\overline{S}$ into $\overline{\overline{S}}$.

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.

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.

Two pieces of this problem:

- How to evaluate the conditional expectation (integral)?

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.

Two pieces of this problem:

- How to evaluate the conditional expectation (integral)?
- How to do constrained optimization (esp. in more than one dimension)?

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.

Two pieces of this problem:

- How to evaluate the conditional expectation (integral)?
- How to do constrained optimization (esp. in more than one dimension)?
- There are quick-and-dirty methods that are slow and inaccurate, and advanced methods that are fast and accurate. To do any kind of ambitious work, you will need the latter.

Simple Analytical Example

Let's Start with a Simple Analytical Example

Neoclassical Growth Model

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

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

Let's Start with a Simple Analytical Example

Neoclassical Growth Model

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

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

- Rewrite the Bellman equation as:

$$V(k) = \max_{c,k'} \{\log (Ak^{\alpha} - k') + \beta V(k')\}$$

Let's Start with a Simple Analytical Example

Neoclassical Growth Model

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

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

- Rewrite the Bellman equation as:

$$V(k) = \max_{c,k'} \{\log (Ak^{\alpha} - k') + \beta V(k')\}$$

- Our goal is to find $V(k)$ and a decision rule g such that $k' = g(k)$

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:

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

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:

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

- $V_1 = \max_{k'} \log(Ak^\alpha - k') \Rightarrow k' = 0 \Rightarrow V_1(k) = \log A + \alpha \log k$

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

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

- $V_1 = \max_{k'} \log\left(Ak^{\alpha} - k'\right) \Rightarrow k' = 0 \Rightarrow V_1(k) = \log A + \alpha \log k$

- Substitute V_1 into the RHS of V_2 :

$$V_2 = \max_{k'} \left\{ \log\left(Ak^{\alpha} - k'\right) + \beta\left(\log A + \alpha \log k'\right) \right\}$$

$$\Rightarrow \text{FOC}: \qquad \frac{1}{Ak^{\alpha} - k'} \quad = \quad \frac{\beta\alpha}{k'} \Rightarrow k' = \frac{\alpha\beta Ak^{\alpha}}{1 + \alpha\beta}$$

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:

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

- $V_1 = \max_{k'} \log\left(Ak^\alpha - k'\right) \Rightarrow k' = 0 \Rightarrow V_1(k) = \log A + \alpha \log k$

- Substitute V_1 into the RHS of V_2 :

$$V_2 = \max_{k'} \left\{\log\left(Ak^\alpha - k'\right) + \beta\left(\log A + \alpha \log k'\right)\right\}$$

$$\Rightarrow \text{FOC}: \quad \frac{1}{Ak^\alpha - k'} = \frac{\beta\alpha}{k'} \Rightarrow k' = \frac{\alpha\beta Ak^\alpha}{1+\alpha\beta}$$

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:

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

- $V_1 = \max_{k'} \log\left(Ak^\alpha - k'\right) \Rightarrow k' = 0 \Rightarrow V_1(k) = \log A + \alpha \log k$

- Substitute V_1 into the RHS of V_2 :

$$V_2 = \max_{k'} \left\{ \log\left(Ak^\alpha - k'\right) + \beta\left(\log A + \alpha \log k'\right) \right\}$$

$$\Rightarrow \text{FOC}: \quad \frac{1}{Ak^\alpha - k'} = \frac{\beta\alpha}{k'} \Rightarrow k' = \frac{\alpha\beta Ak^\alpha}{1 + \alpha\beta}$$

- Substitute k' to obtain V_2. We can keep iterating to find the solution.

II. Guess and Verify (Value Function)

- But there is a more direct approach.

II. Guess and Verify (Value Function)

- But there is a more direct approach.
- Note that both V_2 and V_1 have the same form: $a + b \log k$

II. Guess and Verify (Value Function)

- But there is a more direct approach.
- Note that both V_2 and V_1 have the same form: $a + b \log k$
- Conjecture that the solution $V^*(k) = a + b \log k$, where a and b are coefficients that need to be determined.

$$a + b \log k = \max_{c,k'} \left\{ \log\left(Ak^{\alpha} - k'\right) + \beta\left(a + b \log k'\right) \right\}$$

II. Guess and Verify (Value Function)

- But there is a more direct approach.
- Note that both V_2 and V_1 have the same form: $a + b \log k$
- Conjecture that the solution $V^*(k) = a + b \log k$, where a and b are coefficients that need to be determined.

$$a + b \log k = \max_{c,k'} \{\log (Ak^\alpha - k') + \beta (a + b \log k')\}$$

- FOC:

$$\frac{1}{Ak^\alpha - k'} = \frac{\beta b}{k'} \Rightarrow k' = \frac{\beta b}{1 + \beta b} Ak^\alpha$$

II. Guess and Verify (Value Function)

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

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

II. Guess and Verify (Value Function)

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

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

- Imposing the condition that $LHS \equiv RHS$ for all k, we find a and b :

$$\begin{aligned} a &= \frac{1}{1-\beta}\frac{1}{1-\alpha\beta}\left[\begin{array}{c} \log A + (1-\alpha\beta)\log(1-\alpha\beta) \\ +\alpha\beta\log\alpha\beta \end{array}\right] \\ b &= \frac{\alpha}{1-\alpha\beta} \end{aligned}$$

II. Guess and Verify (Value Function)

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

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

- Imposing the condition that $LHS \equiv RHS$ for all k, we find a and b :

$$
\begin{aligned}
a &= \frac{1}{1-\beta}\frac{1}{1-\alpha\beta}\left[\begin{array}{c}\log A + (1-\alpha\beta)\log(1-\alpha\beta) \\ +\alpha\beta\log\alpha\beta\end{array}\right] \\
b &= \frac{\alpha}{1-\alpha\beta}
\end{aligned}
$$

- We have solved the model!

Guess and Verify as a Numerical Tool

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

Guess and Verify as a Numerical Tool

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

- As long as the true value function is “well-behaved” (smooth, continuous, etc), we can choose a sufficiently flexible family of functions that has a finite (ideally small) number of parameters.

Guess and Verify as a Numerical Tool

- Although this was a very special example, the same general idea underlies many numerical methods:
- As long as the true value function is “well-behaved” (smooth, continuous, etc), we can choose a sufficiently flexible family of functions that has a finite (ideally small) number of parameters.
- Then we can apply the same logic as above and solve for the unknown coefficients, which then gives us the complete solution.

Guess and Verify as a Numerical Tool

- Although this was a very special example, the same general idea underlies many numerical methods:
- As long as the true value function is "well-behaved" (smooth, continuous, etc), we can choose a sufficiently flexible family of functions that has a finite (ideally small) number of parameters.
- Then we can apply the same logic as above and solve for the unknown coefficients, which then gives us the complete solution.
- Many solution methods rely on various versions of this general idea (perturbation methods, collocation methods, parametrized expectations, Krusell-Smith, etc.).

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 $g\left(k\right)$.

III. Guess and Verify (Policy Functions)

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

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

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

- Guess $g(k) = sAk^{\alpha}$, and substitute above:

$$\frac{1}{(1-s)Ak^{\alpha}} = \frac{\beta\alpha A (sAk^{\alpha})^{\alpha-1}}{A\left((sAk^{\alpha})^{\alpha} - sA(aAk^{\alpha})^{\alpha}\right)}$$

III. Guess and Verify (Policy Functions)

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

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

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

- Guess $g(k) = sAk^{\alpha}$, and substitute above:

$$\frac{1}{(1-s)Ak^{\alpha}} = \frac{\beta \alpha A (sAk^{\alpha})^{\alpha-1}}{A\left((sAk^{\alpha})^{\alpha} - sA(aAk^{\alpha})^{\alpha}\right)}$$

- As can be seen, k cancels out, and we get $s = \alpha\beta$.

III. Guess and Verify (Policy Functions)

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

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

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

- Guess $g(k) = sAk^{\alpha}$, and substitute above:

$$\frac{1}{(1-s)Ak^{\alpha}} = \frac{\beta\alpha A\left(sAk^{\alpha}\right)^{\alpha-1}}{A\left(\left(sAk^{\alpha}\right)^{\alpha} - sA\left(aAk^{\alpha}\right)^{\alpha}\right)}$$

- As can be seen, k cancels out, and we get $s = \alpha\beta$.

- By using a very flexible choice of $g()$ this method too can be used for solving very general models.

Numerical Value Function Iteration (VFI)

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 maximizing the right-hand side of the Bellman equation.
3. Stop if convergence criteria satisfied: $|V_{n+1} - V_n| < \texttt{toler}$. Otherwise, increase n and return to step 2.

VFI is Slow. How to Speed It Up?

- VFI can be very slow when $\beta \approx 1$. Three ways to accelerate:

VFI is Slow. How to Speed It Up?

- VFI can be very slow when $\beta \approx 1$. Three ways to accelerate:

  1. (Howard's) **Policy Iteration Algorithm** (together with its "modified" version)

VFI is Slow. How to Speed It Up?

- VFI can be very slow when $\beta \approx 1$. Three ways to accelerate:

  1. (Howard's) **Policy Iteration Algorithm** (together with its "modified" version)
  2. **MacQueen-Porteus** (MQP) error bounds

VFI is Slow. How to Speed It Up?

- VFI can be very slow when $\beta \approx 1$. Three ways to accelerate:

  1. (Howard's) **Policy Iteration Algorithm** (together with its "modified" version)
  2. **MacQueen-Porteus** (MQP) error bounds
  3. **Endogenous Grid Method** (EGM).

VFI is Slow. How to Speed It Up?

- VFI can be very slow when $\beta \approx 1$. Three ways to accelerate:

  1. (Howard's) **Policy Iteration Algorithm** (together with its "modified" version)
  2. **MacQueen-Porteus** (MQP) error bounds
  3. **Endogenous Grid Method** (EGM).

- In general, basic VFI should never be used without at least one of these add-ons.
  - **EGM is your best bet when it's applicable.** But in certain cases, it's not.
  - In those cases, a combination of Howard's algorithm and MQP can be very useful.

Consider the neoclassical growth model:

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

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

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

Howard's Policy Iteration

Consider the neoclassical growth model:

$$V(k,z) = \max_{c,k'} \left\{ \frac{c^{1-\gamma}}{1-\gamma} + \beta \mathbb{E}\left(V(k',z')\,|z\right) \right\}$$
$$\text{s.t} \quad c + k' = e^{z} k^{\alpha} + (1-\delta)k \tag{P1}$$
$$z' = \rho z + \eta', \qquad\qquad k' \geq \underline{k}.$$

- In stage n of the VFI algorithm, **first**, we maximize the RHS and solve for the policy rule:

$$\tilde{s}_n(k,z) = \arg\max_{s \geq \underline{k}} \left\{ \frac{\left(e^{z_j} k_i^{\alpha} + (1-\delta)k - s\right)^{1-\gamma}}{1-\gamma} + \beta \mathbb{E}\left(V_n(s,z')\,|z\right) \right\}. \tag{1}$$

- **Second**: Plug $\tilde{s}_n(k,z)$ into eq. (1), which I will call "**Howard's mapping**":

$$V_{n+1} = T_{\tilde{s}_n} V_n. \tag{2}$$

Policy Iteration

- Maximization step can be time consuming. So it seems like a waste to use the new policy for only one period in updating to V_{n+1}.

Policy Iteration

- Maximization step can be time consuming. So it seems like a waste to use the new policy for only one period in updating to V_{n+1}.

- A simple but key insight is that $T_{\tilde{s}_n}$ (in eq. 2) is also a contraction mapping with modulus β.
  $\longrightarrow$ if we apply $T_{\tilde{s}_n}$ repeatedly, it also converges to a fixed point itself at rate β.

Policy Iteration

- Maximization step can be time consuming. So it seems like a waste to use the new policy for only one period in updating to V_{n+1}.

- A simple but key insight is that $T_{\tilde{s}_n}$ (in eq. 2) is also a contraction mapping with modulus β.
  $\longrightarrow$ if we apply $T_{\tilde{s}_n}$ repeatedly, it also converges to a fixed point itself at rate β.

- Of course, this fixed point is not the solution of the original Bellman equation we would like to solve.

Policy Iteration

- Maximization step can be time consuming. So it seems like a waste to use the new policy for only one period in updating to V_{n+1}.
- A simple but key insight is that $T_{\tilde{s}_n}$ (in eq. 2) is also a contraction mapping with modulus β.
  $\rightarrow$ if we apply $T_{\tilde{s}_n}$ repeatedly, it also converges to a fixed point itself at rate β.
- Of course, this fixed point is not the solution of the original Bellman equation we would like to solve.
- But it is an operator that is much cheaper to apply. So we may want to apply it more than once.

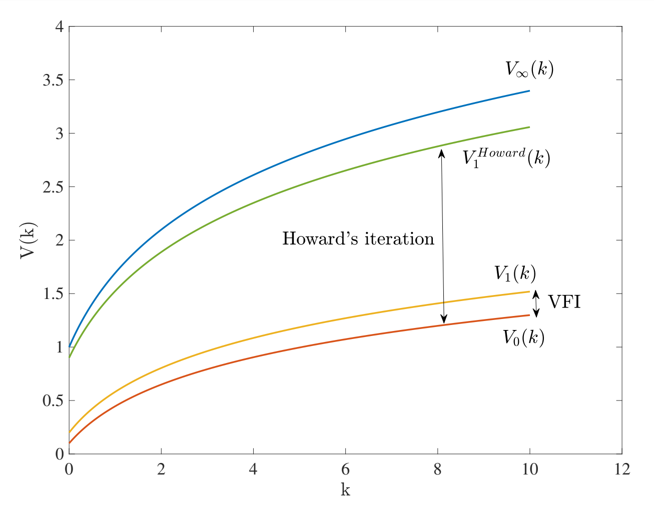
Algorithmus 2 : VFI WITH POLICY ITERATION ALGORITHM

1. Set $n = 0$. Choose an initial guess $V_0 \in S$.

2. Obtain $\tilde{s}_n$ as in (1) and take the updated value function to be: $V_{n+1} = \lim_{m\to\infty} T^m_{\tilde{s}_n} V_n$, which is the (fixed point) value function resulting from using policy $\tilde{s}_n$ forever.

3. Stop if convergence criteria satisfied: $|V_{n+1} - V_n| < \texttt{toler}$. Otherwise, increase n and return to step 1.

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.

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.
- Thus, it inherits two properties of Newton's method:
  1. it is guaranteed to converge to the true solution when the initial point, V_0, is in *the domain of attraction* of V^*, and
  2. when (i) is satisfied, it converges at a *quadratic rate* in iteration index n.

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.
- Thus, it inherits two properties of Newton's method:
  1. it is guaranteed to converge to the true solution when the initial point, V_0, is in *the domain of attraction* of V^*, and
  2. when (i) is satisfied, it converges at a *quadratic rate* in iteration index n.
- Bad news: no more global convergence like VFI (unless state space is discrete)

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.
- Thus, it inherits two properties of Newton's method:
  1. it is guaranteed to converge to the true solution when the initial point, V_0, is in *the domain of attraction* of V^*, and
  2. when (i) is satisfied, it converges at a *quadratic rate* in iteration index n.
- Bad news: no more global convergence like VFI (unless state space is discrete)
- Good news: potentially very fast convergence.

Modified Policy Iteration

Caution:

- Quadratic convergence is a bit misleading: this is the rate in n.

Modified Policy Iteration

Caution:

- Quadratic convergence is a bit misleading: this is the rate in *n*.
  - In contrast to VFI, Howard's algorithm takes a lot of time to evaluate step 2.
- So overall, it may not be much faster when the state space is large and if m is too large.

Modified Policy Iteration

Caution:

- Quadratic convergence is a bit misleading: this is the rate in n.
  - In contrast to VFI, Howard's algorithm takes a lot of time to evaluate step 2.
- So overall, it may not be much faster when the state space is large and if m is too large.
- Second, the basin of attraction can be small.
  - Your algorithm can keep crashing!

Modified Policy Iteration

Caution:

- Quadratic convergence is a bit misleading: this is the rate in n.
  - In contrast to VFI, Howard's algorithm takes a lot of time to evaluate step 2.
- So overall, it may not be much faster when the state space is large and if m is too large.
- Second, the basin of attraction can be small.
  - Your algorithm can keep crashing!
- These can be fixed by slightly **modifying the algorithm.**

VFI with Modified Policy Iteration Algorithm

- Modify *Step 2* of Howard's algorithm:
  - Obtain $\tilde{s}_n$ as in (1) and update the value function to be: $V_{n+1} = T_{\tilde{s}_n}^{m} V_n$, which entails m applications of Howard's mapping to obtain V_{n+1}.

VFI with Modified Policy Iteration Algorithm

- Modify *Step* 2 of Howard's algorithm:
  - Obtain $\tilde{s}_n$ as in (1) and update the value function to be: $V_{n+1} = T^{m}_{\tilde{s}_n} V_n$, which entails m applications of Howard's mapping to obtain V_{n+1}.
- The choice of m will be a key decision to make.
  - HW #1 asks you to experiment to see the tradeoffs.
  - We will also see some benchmarking results in Lecture 4 to help guide this choice.

VFI with Modified Policy Iteration Algorithm

- Modify *Step* 2 of Howard's algorithm:
  - Obtain $\tilde{s}_n$ as in (1) and update the value function to be: $V_{n+1} = T^m_{\tilde{s}_n} V_n$, which entails m applications of Howard's mapping to obtain V_{n+1}.
- The choice of m will be a key decision to make.
  - HW #1 asks you to experiment to see the tradeoffs.
  - We will also see some benchmarking results in Lecture 4 to help guide this choice.
- Note: In some cases we will see later, the iteration will be unstable or will not converge smoothly. In such cases, it will be optimal to **slow down** (or **dampen**) rather than accelerate the Bellman iteration (effectively $m < 1$). This is how $\longrightarrow$

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.)

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.)

3*. Obtain V_{n+1} by taking a convex combination of J_{n+1} and V_n :

$$V_{n+1} = \theta J_{n+1} + (1 - \theta) V_n \quad \text{with} \quad \theta \in (0, 1].$$

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.)

3*. Obtain V_{n+1} by taking a convex combination of J_{n+1} and V_n :

$$V_{n+1} = \theta J_{n+1} + (1-\theta)V_n \quad \text{with} \quad \theta \in (0,1].$$

4*. Stop if convergence criteria satisfied: $|V_{n+1} - V_n| < \texttt{toler}$. Otherwise, increase n and return to step 1.

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.)

3*. Obtain V_{n+1} by taking a convex combination of J_{n+1} and V_n :

$$V_{n+1} = \theta J_{n+1} + (1 - \theta) V_n \quad \text{with} \quad \theta \in (0, 1].$$

4*. Stop if convergence criteria satisfied: $|V_{n+1} - V_n| < \texttt{toler}$. Otherwise, increase n and return to step 1.

- Note: VFI corresponds to $\theta = 1$.

MacQueen-Porteus Bounds

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.

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.
- Contraction mapping theorem can be used to show:

$$\|V^* - V_k\|_\infty \leq \frac{1}{1-\beta} \|V_{k+1} - V_k\|_\infty .$$

- So if we want to stop when the value function is ε away from the true solution, our stopping criterion is:

$$\|V_{k+1} - V_k\|_\infty < \varepsilon \times (1 - \beta).$$

Two Remarks

1. This 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.

Two Remarks

1. This 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.
2. Another issue is how to choose ε. Deviation in V space does not have a natural mapping into economic magnitudes we care about since V does not have a natural scale.

Two Remarks

1. This 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.
2. Another issue is how to choose ε. Deviation in V space does not have a natural mapping into economic magnitudes we care about since V does not have a natural scale.

- One way to address both issues is by defining the stopping rule in the policy function space:
   - It is typically easier to judge what it means to consume or save x% less than optimal (caution: we will see exceptions!)
   - **Also:** Policy functions converge faster than values, so this typically allows stopping sooner.

MacQueen-Porteus Bounds

Consider a different formulation for a dynamic programming problem:

$$V(x_i) = \max_{y \in \Gamma(x_i)} \left[U(x_i, y) + \beta \sum_{j=1}^{J} \pi_{ij}(y) V(x_j) \right], \tag{3}$$

- 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*

$$V(x_i) = \max_{y \in \Gamma(x_i)} \left[U(x_i, y) + \beta \sum_{j=1}^{J} \pi_{ij}(y) V(x_j) \right], \tag{4}$$

define

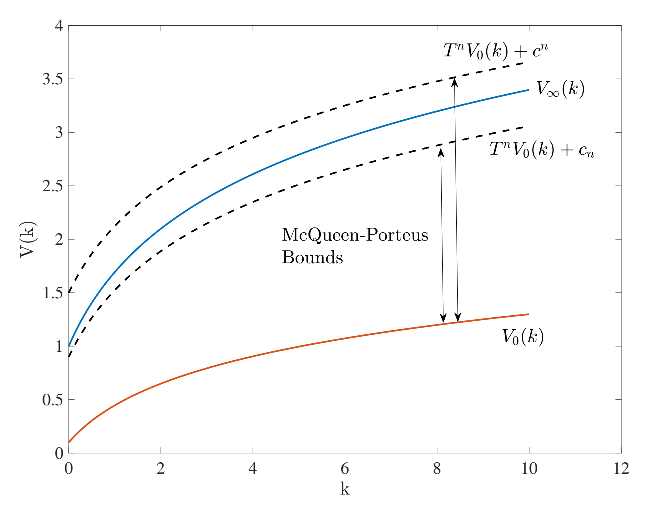
$$\underline{c}_n = \frac{\beta}{1-\beta} \times \min \left[V_n - V_{n-1} \right] \qquad \bar{c}_n = \frac{\beta}{1-\beta} \times \max \left[V_n - V_{n-1} \right] \tag{5}$$

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

$$T^n V_0(\bar{x}) + \underline{c}_n \leq V^*(\bar{x}) \leq T^n V_0(\bar{x}) + \bar{c}_n. \tag{6}$$

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 $V_n(\bar{x}) - V_{n-1}(\bar{x}) = \alpha$ for all $\bar{x}$ and that $\alpha = 100$ (a large number).

MQP Bounds: Comments

- MQP bounds can be quite tight.
- Example: Suppose $V_n(\bar{x}) - V_{n-1}(\bar{x}) = \alpha$ for all $\bar{x}$ and that $\alpha = 100$ (a large number).
- The usual bound implies: $\|V^* - V_n\|_\infty \leq \frac{1}{1-\beta}\left\|V_n(\bar{x}) - V_{n-1}(\bar{x})\right\|_\infty = \frac{\alpha}{1-\beta}$, so we would keep iterating.
- MQP implies $\underline{c}_n = \bar{c}_n = \alpha$, which the then implies

$$\frac{\alpha\beta}{1-\beta} = V^*(\bar{x}) - T^n V_0(\bar{x}) = \frac{\alpha\beta}{1-\beta}.$$

MQP Bounds: Comments

- MQP bounds can be quite tight.
- Example: Suppose $V_n(\bar{x}) - V_{n-1}(\bar{x}) = \alpha$ for all $\bar{x}$ and that $\alpha = 100$ (a large number).
- The usual bound implies: $\|V^* - V_n\|_\infty \leq \frac{1}{1-\beta} \left\|V_n(\bar{x}) - V_{n-1}(\bar{x})\right\|_\infty = \frac{\alpha}{1-\beta}$, so we would keep iterating.
- MQP implies $\underline{c}_n = \bar{c}_n = \alpha$, which the then implies

$$\frac{\alpha\beta}{1-\beta} = V^*(\bar{x}) - T^n V_0(\bar{x}) = \frac{\alpha\beta}{1-\beta}.$$

- We find $V^*(\bar{x}) = V_n(\bar{x}) + \frac{\alpha\beta}{1-\beta}$, in one step!
- MQP: both lower and upper bound for signed difference.

VFI Acceleration with MacQueen-Porteus Bounds

Algorithmus 3 : VFI WITH MACQUEEN-PORTEUS ERROR BOUNDS

[*Step 2'*:] Stop when $\bar{c}_n - \underline{c}_n <$ `toler`. Then take the final estimate of V^* to be either the median

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

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

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

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).

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).
- VFI is proportional to the dominant eigenvalue, which is always 1. Multiplied by β, gives convergence rate.

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).

- VFI is proportional to the dominant eigenvalue, which is always 1. Multiplied by β, gives convergence rate.

- Subdominant (2nd largest) eigenvalue ($|\lambda_2|$) is sometimes $\ll 1$ and sometimes not:
  - AR(1) process, discretized: $|\lambda_2| = \rho$ (persistence parameter)
  - More than 1 ergodic set: $|\lambda_2| = 1$.

- When persistence is low, this can lead to substantial improvements in speed.

Benchmarking MQP and PI

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*	33.29	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	1.45	1.00*
yes	0.67	0.67	0.69	0.14	0.24	0.30	0.02	0.04	0.06

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

Takeaways from the Example

1. Relative to plain VFI ($m = 0$):
   - Modified Howard algorithm alone speeds up by 13 to 33 times
   - MQP speeds up by 19 to 3300 times.
2. Both algorithms most useful when β is high (which is a robust conclusion)
3. The two algorithms are not additive or even always complements:
   - When MQP is used, adding Howard's iteration *slows down* the solution (notice rising times in second rows)
   - When Howard is used, MQP still speeds up solution but less than before: by as low as 1.5 fold for $\beta = 0.95$ but as high as 25 fold for higher β.

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.
- For example:
  - In GE models, especially with more than one asset (or price) or other challenging features, using high Howard iterations early on may cause the algorithm to crash.
  - In practice, I have used m values as high as 20 or even 50 in simpler problems and lower in more complex ones (and often $m < 1$ early in GE iterations!).
  - Be cautious and experiment until you find the sweet spot.
- To sum up, when EGM is not feasible, a combination of Howard and MQP is a good default to use.
- Even with EGM, MQP and Howard can help further speed up the code.

Endogenous Grid Method

Endogenous Grid Method

- In standard VFI, we have

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

Endogenous Grid Method

- In standard VFI, we have

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

- This equation can be rewritten (by substituting out consumption using the budget constraint) as

$$\left(z_j k_i{}^{\alpha} + (1-\delta) k_i - k'\right)^{-\gamma} = \beta \mathbb{E}\left(V_k\left(k', z'\right)|z_j\right), \tag{7}$$

Endogenous Grid Method

- In standard VFI, we have

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

- This equation can be rewritten (by substituting out consumption using the budget constraint) as

$$\left(z_j k_i^{\alpha} + (1 - \delta) k_i - k' \right)^{-\gamma} = \beta \mathbb{E} \left(V_k \left(k', z' \right) | z_j \right), \tag{7}$$

- In VFI, we solve for k' for each grid point today (k_i, z_j).

Endogenous Grid Method

- In standard VFI, we have

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

- This equation can be rewritten (by substituting out consumption using the budget constraint) as

$$\left(z_j k_i^{\alpha} + (1-\delta) k_i - k'\right)^{-\gamma} = \beta \mathbb{E}\left(V_k\left(k', z'\right)|z_j\right), \tag{7}$$

- In VFI, we solve for k' for each grid point today (k_i, z_j).
- Slow for three reasons:

Endogenous Grid Method

- In standard VFI, we have

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

- This equation can be rewritten (by substituting out consumption using the budget constraint) as

$$\left(z_j k_i{}^{\alpha} + (1-\delta) k_i - k'\right)^{-\gamma} = \beta \mathbb{E}\left(V_k\left(k', z'\right)|z_j\right), \tag{7}$$

- In VFI, we solve for k' for each grid point today (k_i, z_j).
- Slow for three reasons:
  1. This is a *non-linear* equation in k'.

Endogenous Grid Method

- In standard VFI, we have

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

- This equation can be rewritten (by substituting out consumption using the budget constraint) as

$$\left(z_j k_i^{\alpha} + (1-\delta)k_i - k'\right)^{-\gamma} = \beta \mathbb{E}\left(V_k\left(k', z'\right)|z_j\right), \tag{7}$$

- In VFI, we solve for k' for each grid point today (k_i, z_j).
- Slow for three reasons:
  1. This is a *non-linear* equation in k'.
  2. $V(k_i, z_j)$ is stored at grid points, so for every trial value of k', we need to:
     - 2.1 evaluate the conditional expectation (since k' appears inside the expectation), and
     - 2.2 interpolate to obtain off-grid values $V(k', z'_j)$ for each z'_j.

EGM

- View the problem differently:

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

EGM

- View the problem differently:

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

$$\text{s.t} \quad c + k_i' = z_j k^{\alpha} + (1-\delta)k \tag{P3}$$

$$\ln z' = \rho \ln z_j + \eta',$$

- Now the same FOC as before:

$$\left(z_j k^{\alpha} + (1-\delta)k - k_i'\right)^{-\gamma} = \beta \mathbb{E}\left(V_k\left(k_i', z'\right)|z_j\right), \tag{8}$$

but solve for k as a function of k_i' and z_j :

$$z_j k^{\alpha} + (1-\delta)k = \left[\beta \mathbb{E}\left(V_k\left(k_i', z'\right)|z_j\right)\right]^{-1/\gamma} + k_i'.$$

EGM

- View the problem differently:

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

- Now the same FOC as before:

$$\left(z_j k^\alpha + (1-\delta)k - k'_i\right)^{-\gamma} = \beta \mathbb{E}\left(V_k\left(k'_i, z'\right) | z_j\right), \tag{8}$$

but solve for k as a function of k'_i and z_j :

$$z_j k^\alpha + (1-\delta)k = \left[\beta \mathbb{E}\left(V_k\left(k'_i, z'\right) | z_j\right)\right]^{-1/\gamma} + k'_i.$$

- Trick 1: RHS is now entirely on the (k'_i, z_j) grid. So, no need to interpolate/integrate RHS repeatedly as before! (Solve problems 2.1, 2.2 above).

EGM

- View the problem differently:

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

- Now the same FOC as before:

$$\left(z_j k^\alpha + (1-\delta)k - k'_i\right)^{-\gamma} = \beta \mathbb{E}\left(V_k\left(k'_i, z'\right) | z_j\right), \tag{8}$$

but solve for k as a function of k'_i and z_j :

$$z_j k^\alpha + (1-\delta)k = \left[\beta \mathbb{E}\left(V_k\left(k'_i, z'\right) | z_j\right)\right]^{-1/\gamma} + k'_i.$$

- Trick 1: RHS is now entirely on the (k'_i, z_j) grid. So, no need to interpolate/integrate RHS repeatedly as before! (Solve problems 2.1, 2.2 above).
- Problem 1 still remains: LHS still nonlinear in k.

- Trick 2: Define
$$Y \equiv zk^{\alpha} + (1-\delta)k \tag{9}$$
and rewrite the Bellman equation (without discretization) as:
$$\mathcal{V}(Y,z) = \max_{k'} \left\{ \frac{(Y-k')^{1-\gamma}}{1-\gamma} + \beta \mathbb{E}\left(\mathcal{V}(Y',z') \,|z\right) \right\}$$
$$\text{s.t} \qquad \ln z' = \rho \ln z + \eta'.$$

- <2->Key observation: Y' is only a function of k_i' and z', so we can write the conditional expectation on the right hand side as:
$$\mathbb{V}(k_i', z_j) \equiv \beta \mathbb{E}\left(\mathcal{V}\left(Y'(k_i', z'), z'\right) |z_j\right).$$

EGM

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

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

EGM

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

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

- Now the FOC of this new problem becomes:

$$c^*(k'_i, z_j)^{-\gamma} = \mathbb{V}_{k'}(k'_i, z_j). \tag{10}$$

EGM

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

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

- Now the FOC of this new problem becomes:

$$c^*(k'_i, z_j)^{-\gamma} = \mathbb{V}_{k'}(k'_i, z_j). \tag{10}$$

- Magic! This equation gives us consumption in one step:
  - without searching over values of k'—hence avoiding repeated interpolation and integration!
  - without solving a nonlinear equation in k'

EGM

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

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

- Now the FOC of this new problem becomes:

$$c^*(k'_i,z_j)^{-\gamma}=\mathbb{V}_{k'}(k'_i,z_j). \tag{10}$$

- Magic! This equation gives us consumption in one step:
  - without searching over values of k'—hence avoiding repeated interpolation and integration!
  - without solving a nonlinear equation in k'
- Once $c^*(k'_i,z_j)$ is obtained, use the resource constraint to compute today's end-of-period resources: $Y^*(k'_i,z_j)=c^*(k'_i,z_j)+k'_i$ as well as

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

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

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

1: For all i, j, obtain

$$c^*(k'_i, z_j) = \left(\mathbb{V}^n_k(k'_i, z_j)\right)^{-1/\gamma}.$$

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 $\mathbb{V}^0(k'_i, z_j)$.

1: For all i, j, obtain

$$c^*(k'_i, z_j) = \left(\mathbb{V}^n_k(k'_i, z_j)\right)^{-1/\gamma}.$$

2: Obtain today's end-of-period resources as a function of tomorrow's capital and today's shock:

$$Y^*(k'_i, z_j) = c^*(k'_i, z_j) + k'_i,$$

and today's updated value function,

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

by plugging in consumption decision into the RHS.

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: $Y' = z'(k'_i)^{\alpha} + (1-\delta)k'_i$.

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

4: Obtain

$$\mathbb{V}^{n+1}(k_i', z_j) = \beta \mathbb{E}\left(\mathcal{V}^{n+1}\left(Y'(k_i', z'), z'\right) |z_j\right).$$

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

4: Obtain

$$\mathbb{V}^{n+1}(k'_i, z_j) = \beta\mathbb{E}\left(\mathcal{V}^{n+1}\left(Y'(k'_i, z'), z'\right)|z_j\right).$$

5: Stop if convergence criterion is satisfied and obtain beginning-of-period capital, k, by solving the nonlinear equation $Y^{n*}(i,j) \equiv z_j k^{\alpha} + (1-\delta)k$, for all i, j. Otherwise, go to step 1.

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.