PhD Macro Core Part I:
Lecture 6 – Dynamic Programming II

Min Fang
University of Florida

Fall 2024

Today

- Rough sketch of some important mathematical background
- The idea of contraction mapping and contraction mapping theorem
- Practical dynamic programming
- A simple value function iteration scheme implemented in Matlab

Our Goal

- To solve dynamic programming problems, want to find fixed points

$$v = Tv$$

where T is an operator like

$$Tv(k) \equiv \max_{0 \leqslant x \leqslant f(k)} [u(f(k) - x) + \beta v(x)]$$

- Our approach: viewing functions v as elements in an abstract vector space

- Find the convergence properties of sequences v_n of such functions induced by T

$$v_{n+1}(k) = Tv_n(k)$$

Math Prelim

- Vector spaces and metric spaces
  - We view functions v as elements in a suitably chosen vector space S
  - Need to know whether a sequence v_n of such functions *converges*
  - Need to know the *distance* d (defined in metric spaces (S, d)) between functions

- The idea of convergence of a sequence
  - A sequence $\{v_n\}_{n=0}^{\infty}$ converges to $v \in S$ if for any $\varepsilon > 0$ there exists an N_ε such that

$$d(v_n, v) < \varepsilon \quad \text{for all } n \geqslant N_\varepsilon$$

    REMARK: in other words, the sequence $\{v_n\}$ convergences to $v \in S$ if the sequence of real numbers $\{d(v_n, v)\}$ converges to zero.

Contraction Mapping Theorem

- The idea of the contraction mapping
  - Let (S, d) be a metric space and let $T : S \to S$.
  - Then T is a contraction mapping (with modulus β) if for some $\beta \in (0, 1)$

    $$d(Tx, Ty) \leqslant \beta d(x, y), \qquad \text{for all } x, y \in S$$

    REMARK: in other words, applying T brings x and y closer together.

- **Contraction Mapping Theorem** (Monotonicity + Discounting)
  - Let (S, d) be a complete metric space and let $T : S \to S$ be a contraction mapping.
  - Then T has a unique fixed point $x = Tx$ in S.
  - This is sometimes known as the *Banach* fixed point theorem.

Back to Optimal Growth

- Consider our usual Bellman operator:

$$Tv(k) \equiv \max_x [u(f(k) - x) + \beta v(x)]$$

  - (i) Monotonicity. If $v \leqslant w$ then

    $$u(f(k) - x) + \beta v(x) \leqslant u(f(k) - x) + \beta w(x), \qquad \text{for all } x$$

    so

    $$\max_x [u(f(k) - x) + \beta v(x)] \leqslant \max_x [u(f(k) - x) + \beta w(x)]$$

    so

    $$Tv(k) \leqslant Tw(k)$$

  - Hence T satisfies the monotonicity property.

Back to Optimal Growth

- Consider our usual Bellman operator:

$$Tv(k) \equiv \max_x [u(f(k) - x) + \beta v(x)]$$

  - (ii) Discounting. For any $a \geqslant 0$ we have

$$\begin{aligned} T(v+a)(k) &= \max_x [u(f(k) - x) + \beta(v(x) + a)] \\ &= \max_x [u(f(k) - x) + \beta v(x)] + \beta a \\ &= Tv(k) + \beta a \end{aligned}$$

- So, we are confident that optimal growth could be solved!

Back to Optimal Growth

- Consider our usual Bellman operator:

$$Tv(k) \equiv \max_x [u(f(k) - x) + \beta v(x)]$$

  - (ii) Discounting. For any $a \geqslant 0$ we have

$$\begin{aligned} T(v+a)(k) &= \max_x [u(f(k) - x) + \beta(v(x) + a)] \\ &= \max_x [u(f(k) - x) + \beta v(x)] + \beta a \\ &= Tv(k) + \beta a \end{aligned}$$

- So, we are confident that optimal growth could be solved!

Practical Dynamic Programming

- Suppose we want to solve the Bellman equation for the optimal growth model

$$v(k) = \max_{x \in \Gamma(k)} [u(f(k) - x) + \beta v(x)] \quad \text{for all } k \in \mathcal{K}$$

  where x denotes the capital stock chosen for the next period

- For this problem, the givens are
  - state space $\mathcal{K}$
  - strictly increasing strictly concave production function $f(k)$
  - strictly increasing strictly concave utility function $u(c)$
  - constraint sets of the form $\Gamma(k) = [0, f(k)]$ for each $k \in \mathcal{K}$
  - time discount factor $0 < \beta < 1$

Discrete State Space Approximation

- Now suppose we approximate the continuous state space $\mathcal{K}$ with a suitably chosen finite grid of possible capital stocks

  $$k_{\min} < \ldots < k_i < \ldots < k_{\max}, \quad i = 1, \ldots, n$$

  That is, a vector of length n

- On this grid of points, the value function is also a finite vector

  $$v\left(k_{\min}\right), \ldots, v\left(k_i\right), \ldots, v\left(k_{\max}\right), \quad i = 1, \ldots, n$$

- Write k_i for a typical element of the grid of capital stocks and $v_i = v\left(k_i\right)$ for a typical element of the value function

Discrete State Space Approximation

- Let c_{ij} denote consumption if current capital is $k = k_i$ and capital chosen for next period is $x = k_j$

  $$c_{ij} = f\left(k_i\right) - k_j, \quad i,j = 1,\ldots,n$$

  We will need to be careful to respect the feasibility constraints

  $$0 \leqslant k_j \leqslant f\left(k_i\right), \quad i,j = 1,\ldots,n$$

- Let u_{ij} denote the flow utility associated with c_{ij}

  $$u_{ij} = u\left(c_{ij}\right), \quad i,j = 1,\ldots,n$$

- So u is an $n \times n$ matrix with rows indicating current capital $k = k_i$ and columns indicating feasible choices for $x = k_j$

Discrete State Space Approximation

- In this notation, our Bellman equation can be written

$$v_i = \max_j \left[u_{ij} + \beta v_j\right], \quad i = 1, \ldots, n$$

- Associated with this is the policy function

$$g_i = \operatorname*{argmax}_j \left[u_{ij} + \beta v_j\right], \quad i = 1, \ldots, n$$

such that $g_i = g\left(k_i\right)$ attains the max given $k = k_i$

Value Function Iteration

- Start with an initial guess v_i^0 and then calculate

$$v_i^1 = Tv_i^0 = \max_j \left[u_{ij} + \beta v_j^0\right], \quad i = 1, \ldots, n$$

  and compute the error

$$\left\|Tv^0 - v^0\right\| = \max_i \left[\left|Tv_i^0 - v_i^0\right|\right]$$

- If this error is less than some pre-specified tolerance $\varepsilon > 0$, stop. Otherwise, update to

$$v_i^2 = Tv_i^1 = \max_j \left[u_{ij} + \beta v_j^1\right], \quad i = 1, \ldots, n$$

Value Function Iteration

- Keep iterating on
$$v_i^{l+1} = Tv_i^l = \max_j \left[u_{ij} + \beta v_j^l\right], \quad i = 1, \ldots, n$$
for iterates $l = 0, 1, 2, \ldots$ until
$$\left\|Tv^l - v^l\right\| = \max_i \left[\left|Tv_i^l - v_i^l\right|\right] < \varepsilon$$

- Since T is a contraction mapping, this will converge

- Implementing value function iteration in MATLAB

Implementing value function iteration in MATLAB

Setup

From Matlab script "*value_function_iteration_example.m*" in LMS

```
%%%%% economic parameters

alpha = 1/3;         %% capital's share in production function
beta  = 0.95;        %% time discount factor
delta = 0.05;        %% depreciation rate
sigma = 1;           %% CRRA (=1/IES)
rho   = (1/beta)-1; %% implied rate of time preference

kstar = (alpha/(rho+delta))^(1/(1-alpha)); %% steady state
kbar  = (1/delta)^(1/(1-alpha));
```

Setup

```
%%%%% numerical parameters

max_iter = 500;         %% maximum number of iterations
tol      = 1e-7;        %% treat numbers smaller than this as zero
penalty  = 10^16;       %% for penalizing constraint violations
```

Setup

```
%%%%% setting up the grid of capital stocks

n         = 1001;      %% number of nodes for k grid
kmin      = tol;       %% effectively zero
kmax      = kbar;      %% effective upper bound on k grid

k         = linspace(kmin, kmax, n); %% linearly spaced
```

May need to choose grid ‘artfully’ . . .

Return function

```
%%%%% return function

c    = zeros(n,n);

for j=1:n,

    c(:,j) = (k.^alpha) + (1-delta)*k - k(j);

end
```

But this leads to infeasible choices ...

Return function: enforcing feasibility

```
%%%%% penalize violations of feasibility constraints

violations = (c<=0);

c = c.*(c>=0) + eps;

if sigma==1,

    u = log(c) - penalty*violations;

else

    u = (1/(1-sigma))*(c.^(1-sigma) - 1) - penalty*violations;

end
```

This will ensure that the solution respects feasibility constraints

Bellman iterations

```
%%%%% now solve Bellman equation by value function iteration

%%%%% initial guess

v = zeros(n,1);

%%%%% iterate on Bellman operator

for i=1:max_iter,
```

For loop needs an ‘end’ — see below

Maximization step

```
%%%%% RHS of Bellman equation

RHS     = u + beta*kron(ones(n,1),v');

%%%%% maximize over this to get Tv

[Tv,argmax] = max(RHS,[],2);

%%%%% policy that attains the maximum

g = k(argmax);
```

RHS is an $n \times n$ matrix with rows indicating current $k = k_i$ and columns indicating feasible next period's capital $x = k_j$

For each row entry i, max is taken along the column entries j of RHS

Check if converged

```
%%%%% check if converged

error = norm(Tv-v,inf);

fprintf('%4i %6.2e \n',[i, error]);

if error<tol, break, end;
```

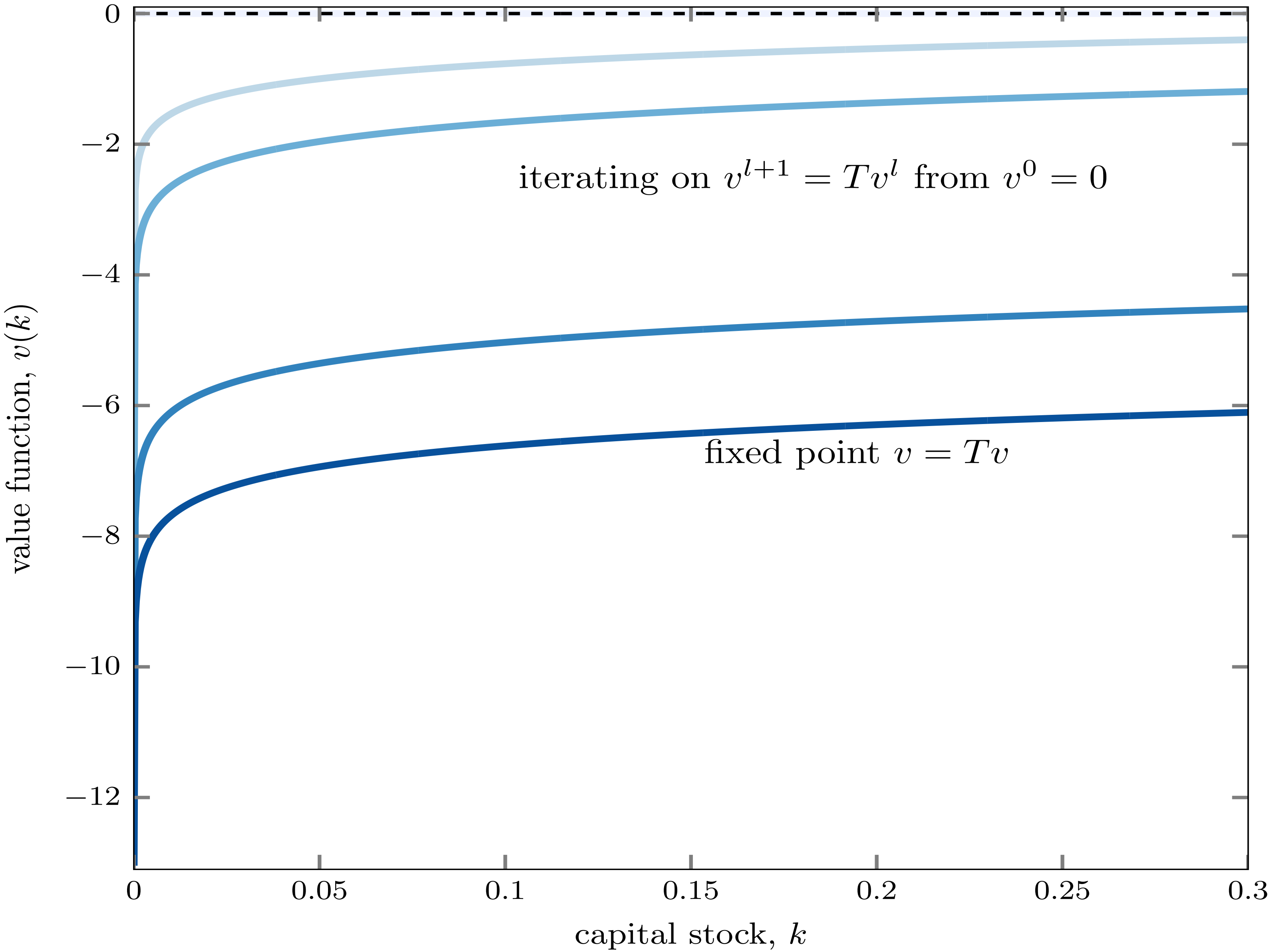
Breaks the for loop if we have error $<$ tolerance

If not, update and try again

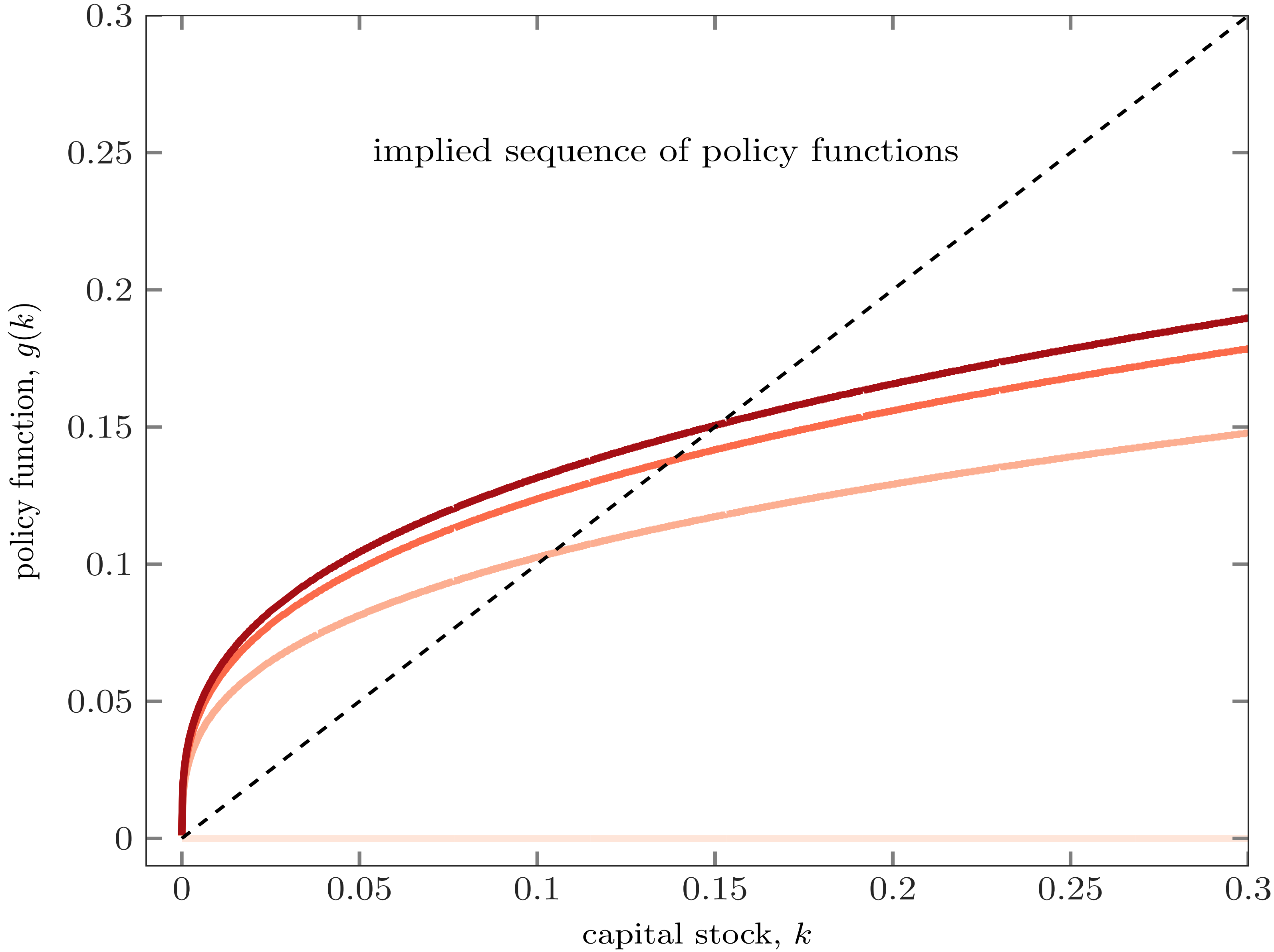
```
%%%%% if not converged, update and try again

v = Tv;

end
```

Here's that end to the for loop, so now we go back to the beginning of the loop but with a new guess at v

Convergence of value functions $v^l \to v = Tv$



Convergence of policy functions $g^l \to g$



Transitional dynamics