Macroeconomics

Lecture 6: dynamic programming methods, part four

Chris Edmond

1st Semester 2019

This class

- Practical dynamic programming
- Crude first approach — *discrete state approximation*
- A simple value function iteration scheme implemented in MATLAB
- Later we'll refine this approach

Practical dynamic programming

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

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

where x denotes the capital stock chosen for 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

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

  That is, a vector of length n

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

$$v(k_{\min}), \ldots, v(k_i), \ldots, v(k_{\max}), \qquad i = 1, ..., n$$

- Write k_i for a typical element of the grid of capital stocks and $v_i = v(k_i)$ 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 $x = k_j$

$$c_{ij} = f(k_i) - k_j, \qquad i, j = 1, ..., n$$

We will need to be careful to respect the feasibility constraints

$$0 \leq k_j \leq f(k_i), \qquad i, j = 1, ..., n$$

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

$$u_{ij} = u(c_{ij}), \qquad i, j = 1, ..., n$$

- 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

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

- Associated to this is the policy function

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

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

Value function iteration

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

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

and compute the *error*

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

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

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

Value function iteration

- Keep iterating on

$$v_i^{l+1} = Tv_i^l = \max_j \left[u_{ij} + \beta v_j^l \right], \qquad i = 1, ..., n$$

for iterates $l = 0, 1, 2, \ldots$ until

$$\|Tv^l - v^l\| = \max_i \left[\left| Tv_i^l - v_i^l \right| \right] < \varepsilon$$

- Since T is a contraction mapping, this will converge

Implementing value function iteration in Matlab

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;
```

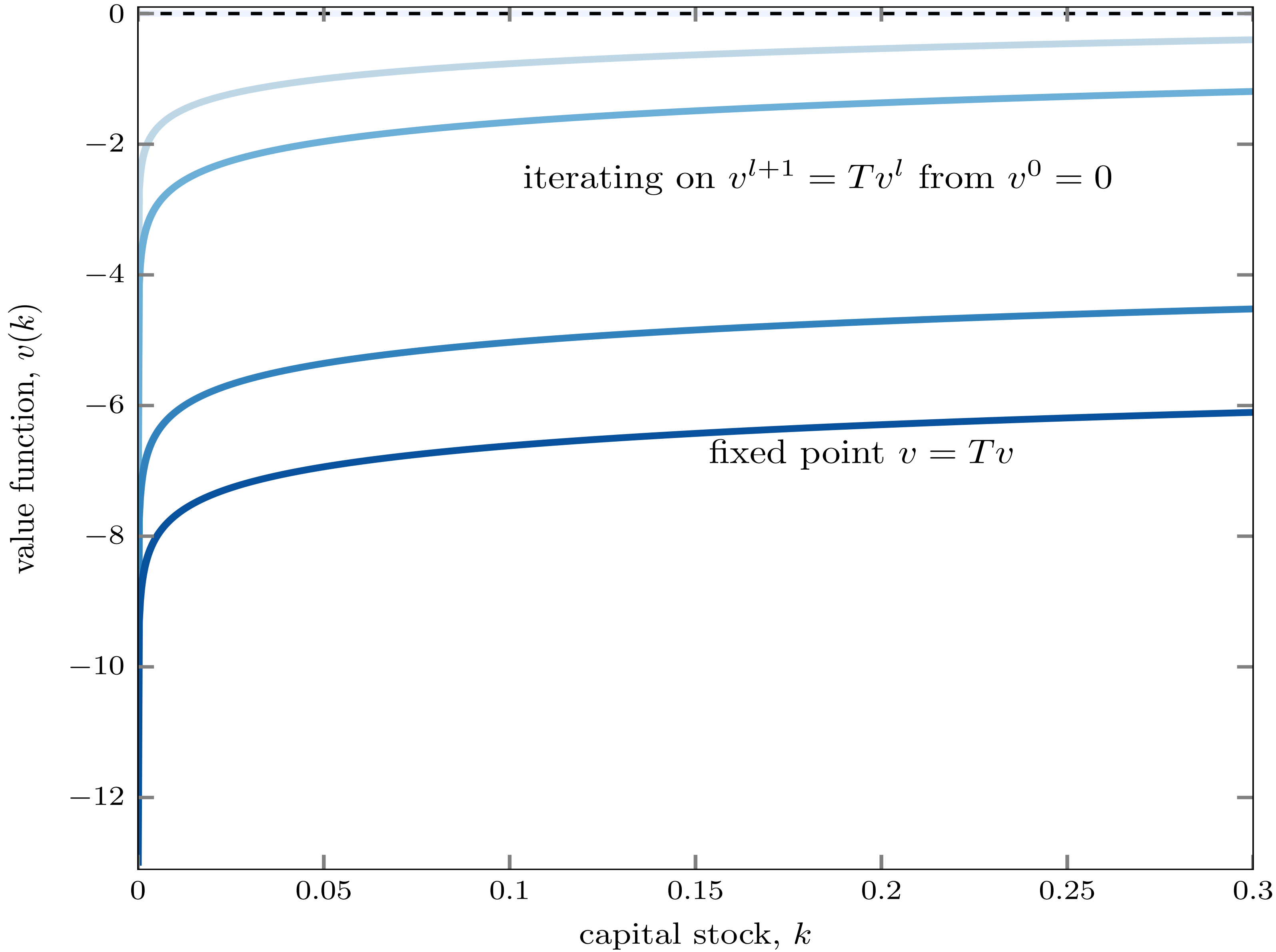
Breaks the for loop if we have error $<$ tolerance

If not, update and try again

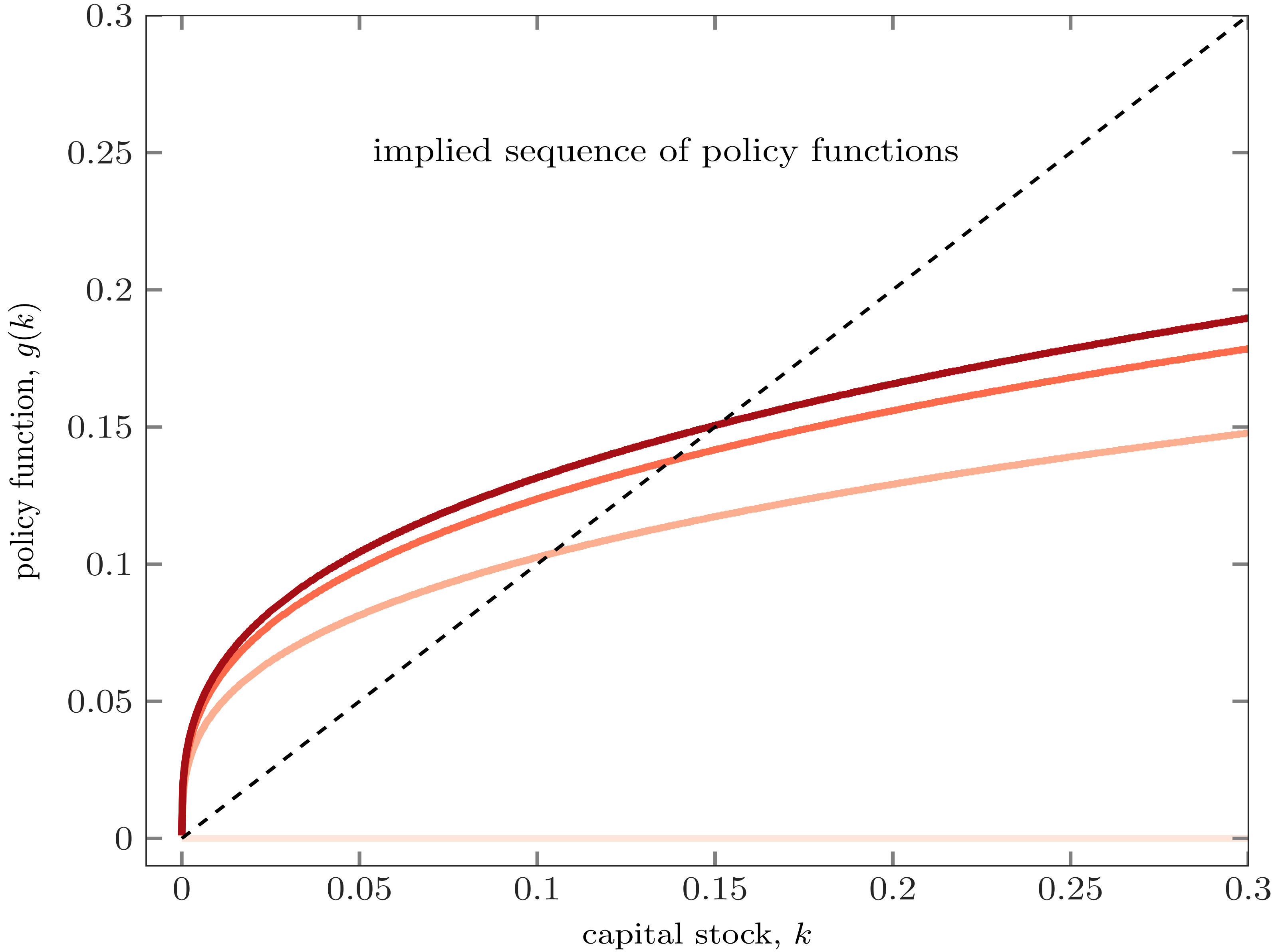
```
%%%%% if not converged, update and try again

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

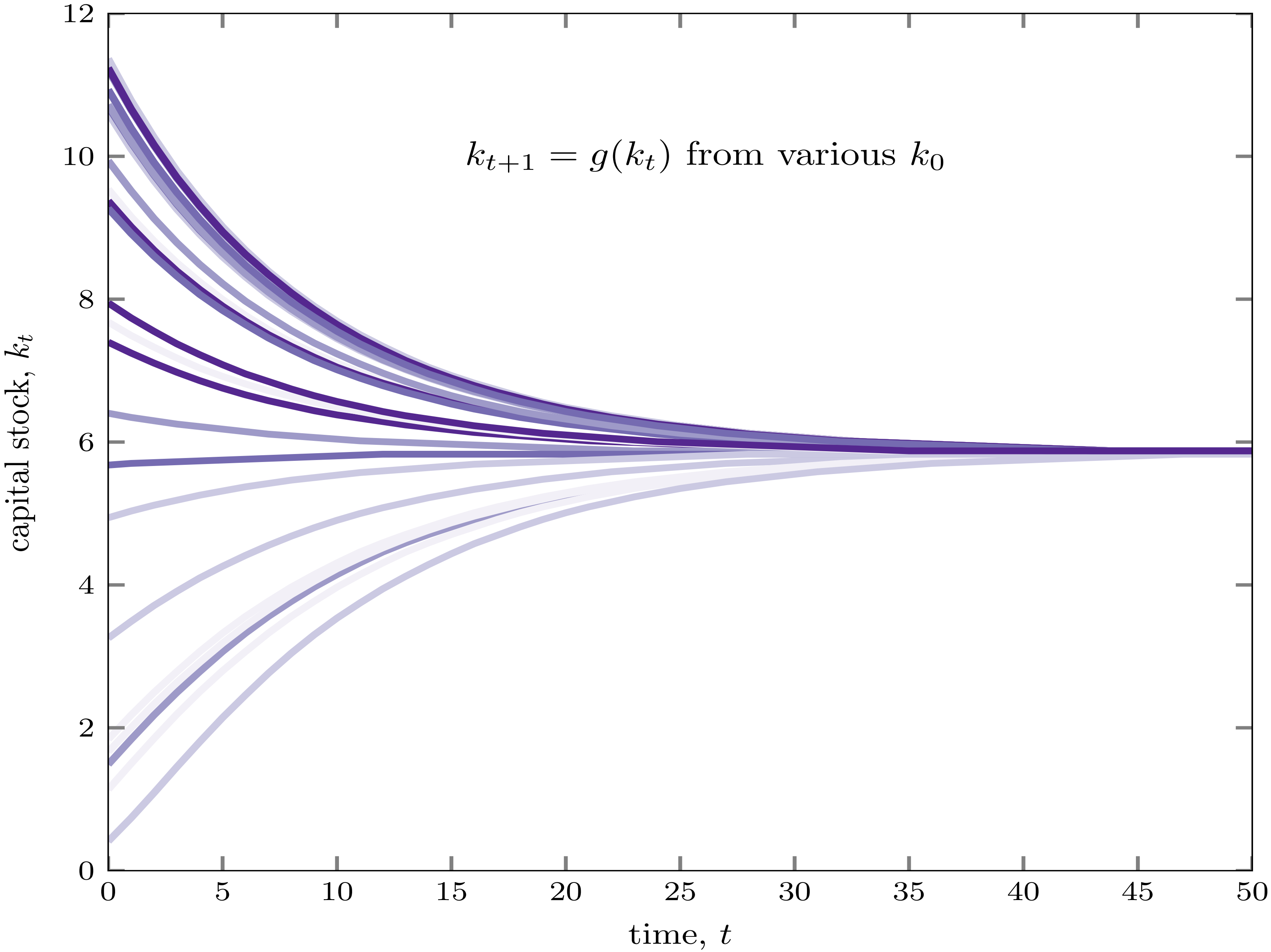
Convergence of value functions $v^l \to v = Tv$



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



Transitional dynamics



Next class

- Refining this approach
- Interpolation and function approximation by collocation