An Introduction to Computational Macroeconomics

Dynamic Programming: Chapter 6

John Stachurski

June – July 2022

Markov Decision Processes

Next we cover Markov decision processes (MDPs)

- A class of dynamic programs
- Broad enough to encompass many economic applications
- Includes optimal stopping problems as a special case
- Clean, powerful theory
- A range of important algorithms

Also a cornerstone for

- reinforcement learning, artificial intelligence, etc.

MDPs are dynamic programs characterized by two features

1. Rewards are additively separable:

$$\text{lifetime reward} \;= \mathbb{E} \sum_{t \geqslant 0} \beta^t R_t$$

2. The discount rate is constant

For now we restrict attention to <u>finite</u> state and action spaces

- Routinely used in quantitative applications
- Avoids technical issues we can put aside for later

Notation

Let X and A be any sets

- $\wp(\mathsf{A}) :=$ the set of all subsets of A

A **correspondence** Γ from X to A is a map from X to $\wp(\mathsf{A})$

- called **nonempty** if $\Gamma(x) \neq \emptyset$ for all $x \in \mathsf{X}$

Examples.

- $\Gamma(x) = [0, x]$ is a correspondence from $\mathbb{R}$ to $\mathbb{R}$
- $\Gamma(x) = [-x, x]$ is a nonempty correspondence from $\mathbb{R}$ to $\mathbb{R}$

We study a controller who, at each integer $t \geqslant 0$

1. observes the current state X_t
2. responds with an action A_t

Her aim is to maximize expected discounted rewards

$$\mathbb{E} \sum_{t \geqslant 0} \beta^t r(X_t, A_t), \qquad X_0 = x_0 \text{ given}$$

We take as given

1. a finite set X called the **state space** and
2. a finite set A called the **action space**

The actions of the controller are limited by a **feasible correspondence** Γ

- A correspondence from X to A
- $\Gamma(x)$ is the set of actions available to the controller in state x

Given Γ, we set

$$\mathsf{G} := \{(x, a) \in \mathsf{X} \times \mathsf{A} : a \in \Gamma(x)\}$$

- called the set of **feasible state-action pairs**

Reward $r(x, a)$ is received at feasible state-action pair (x, a), A **stochastic kernel** from G to X is a map $P \colon \mathsf{G} \times \mathsf{X} \to \mathbb{R}_+$ satisfying

$$\sum_{x' \in \mathsf{X}} P(x, a, x') = 1 \quad \text{for all } (x, a) \text{ in } \mathsf{G}$$

Interpretation

- For each feasible state-action pair, $P(x, a, \cdot)$ is a distribution
- The next period state x' is selected from $P(x, a, \cdot)$

Now let's put it all together:

Given X and A, a **Markov decision process** (**MDP**) is a tuple (Γ, β, r, P) where

1. Γ is a nonempty correspondence from X $\to$ A
2. β is a constant in $(0, 1)$
3. r is a function from G to $\mathbb{R}$
4. P is a stochastic kernel from G to X

In the foregoing,

- β is called the **discount factor**
- r is called the **reward function**

Algorithm 1: MDP dynamics: states, actions, and rewards

$t \leftarrow 0$
input X_0
while $t < \infty$ **do**
  observe X_t
  choose action A_t from $\Gamma(X_t)$
  receive reward $r(X_t, A_t)$
  draw X_{t+1} from $P(X_t, A_t, \cdot)$
  $t \leftarrow t + 1$
end

Rules:

- Choose $(A_t)_{t \geqslant 0}$ to maximize $\mathbb{E} \sum_{t \geqslant 0} \beta^t r(X_t, A_t)$
- Actions don't depend on future outcomes

The **Bellman equation** is

$$v(x) = \max_{a \in \Gamma(x)} \left\{ r(x, a) + \beta \sum_{x' \in \mathsf{X}} v(x') P(x, a, x') \right\}$$

Reduces an infinite horizon problem to a two period problem

In the two period problem, the controller trades off

1. current rewards and
2. expected discounted value from future states

Current actions influence both terms

Example. Cake eating (no labor income), where

$$W_{t+1} = RW_t - C_t \qquad (t = 0, 1, \ldots)$$

- Investing d dollars today returns Rd next period
- $C_t, W_t \geqslant 0$ are current consumption and wealth
- Assume wealth takes values in a finite set $\mathsf{W} \subset \mathbb{R}_+$

The agent seeks to maximize

$$\mathbb{E} \sum_{t \geqslant 0} \beta^t u(C_t) \quad \text{given } W_0 = w$$

Bellman equation:

$$v(w) = \max_{0 \leqslant w' \leqslant Rw} \left\{ u(Rw - w') + \beta v(w') \right\}$$

This model can be framed as an MDP with W as the state space

- The action is $S_t = RW_t - C_t =$ next-period wealth
- The action space is also W
- The feasible correspondence is

$$\Gamma(w) = \{s \in \mathsf{W} : s \leqslant Rw\}$$

- Current reward is

$$r(w, s) = u(Rw - s) \qquad (s \in \Gamma(w))$$

- The stochastic kernel is $P(w, s, w') = \mathbb{1}\{w' = s\}$

Example. All optimal stopping problems can be framed as MDPs

- See the text for details

This is important from a theoretical perspective

- illustrates the generality of MDPs

However, expressing optimal stopping problems as an MDP requires an extra state variable

- status $S_t = 1$ if stopped else zero

Hence treating optimal stopping problems separately is neater

Policies

Actions will be governed by policies

- maps from states to actions
- today's action is a function of today's state!

The set of **feasible policies** is

$$\Sigma := \text{ all } \sigma \in \mathsf{A}^{\mathsf{X}} \text{ s.t. } \sigma(x) \in \Gamma(x) \text{ for all } x \in \mathsf{X}$$

Meaning of selecting σ from Σ:

- respond to state X_t with action $A_t := \sigma(X_t)$ at all t

Dynamics

What happens when we always follow $\sigma \in \Sigma$?

Now

$$X_{t+1} \sim P(X_t, \sigma(X_t), \cdot) \quad \text{at every } t$$

Thus, X_t updates according to the stochastic matrix

$$P_\sigma(x, x') := P(x, \sigma(x), x') \qquad (x, x' \in \mathsf{X})$$

The state process becomes P_σ-Markov

- Fixing a policy "closes the loop" in the state dynamics
- Solving an MDP means choosing a Markov chain!

Rewards

Under the policy σ, rewards at x are $r(x, \sigma(x))$

Let

$$r_\sigma(x) := r(x, \sigma(x)) \qquad (x \in \mathsf{X})$$

Now set

$$\mathbb{E}_x := \mathbb{E}[\,\cdot \mid X_0 = x]$$

Then the expected time t reward is

$$\mathbb{E}_x\, r(X_t, A_t) = \mathbb{E}_x\, r_\sigma(X_t) = (P_\sigma^t\, r_\sigma)(x)$$

Let $(X_t)_{t \geqslant 0}$ be P_σ-Markov with $X_0 = x$

The lifetime value of σ starting from x is

$$v_\sigma(x) := \mathbb{E}_x \sum_{t \geqslant 0} \beta^t r_\sigma(X_t)$$

Since $\beta < 1$, we have $r(\beta P_\sigma) < 1$ and hence

$$v_\sigma = \sum_{t \geqslant 0} \beta^t P_\sigma^t \, r_\sigma = (I - \beta P_\sigma)^{-1} \, r_\sigma$$

The **value function** is defined as

$$v^*(x) = \max_{\sigma \in \Sigma} v_\sigma(x) \qquad (x \in \mathsf{X})$$

Recall that the Bellman equation is

$$v(x) = \max_{a \in \Gamma(x)} \left\{ r(x, a) + \beta \sum_{x' \in \mathsf{X}} v(x') P(x, a, x') \right\}$$

The **Bellman operator** for the MDP is the self-map T on $\mathbb{R}^{\mathsf{X}}$ defined by

$$(Tv)(x) = \max_{a \in \Gamma(x)} \left\{ r(x, a) + \beta \sum_{x' \in \mathsf{X}} v(x') P(x, a, x') \right\}$$

Obviously

- $Tv = v$ iff v satisfies the Bellman equation
- T is order-preserving on $\mathbb{R}^{\mathsf{X}}$

Fix $v \in \mathbb{R}^{\mathsf{X}}$

A policy $\sigma \in \Sigma$ is called **v-greedy** if

$$\forall\, x \in \mathsf{X}, \;\; \sigma(x) \in \operatorname*{argmax}_{a \in \Gamma(x)} \left\{ r(x,a) + \beta \sum_{x' \in \mathsf{X}} v(x') P(x,a,x') \right\}$$

A policy $\sigma \in \Sigma$ is called **optimal** if

$$v_\sigma = v^*$$

Thus,

$$\sigma \text{ is optimal } \iff \text{ lifetime value is maximal at each state}$$

Proposition. For the MDP described above

1. v^* is the unique fixed point of T in $\mathbb{R}^{\mathsf{X}}$
2. T is a contraction of modulus β on $\mathbb{R}^{\mathsf{X}}$ under the norm $\| \cdot \|_\infty$
3. A feasible policy is optimal if and only it is v^*-greedy
4. At least one optimal policy exists

Proof:

- similar to that for optimal stopping
- full details deferred until we study RDPs

Ex. Show that optimal iff v^*-greedy implies that at least one optimal policy exists

Proof: Given $v \in \mathbb{R}^X$, a v-greedy policy is a $\sigma \in \Sigma$ such that

$$\sigma(x) \in \operatorname*{argmax}_{a \in \Gamma(x)} \left\{ r(x,a) + \beta \sum_{x' \in X} v(x') P(x,a,x') \right\} \quad \text{for all } x \in X$$

For each $v \in \mathbb{R}^X$, a v-greedy policy exists

- just select a maximizer a_x^* from the nonempty set $\Gamma(x)$ at each x in X

Hence a v^*-greedy policy exists

Hence an optimal policy exists

Ex. Show that optimal iff v^*-greedy implies that at least one optimal policy exists

Proof: Given $v \in \mathbb{R}^{\mathsf{X}}$, a v-greedy policy is a $\sigma \in \Sigma$ such that

$$\sigma(x) \in \operatorname*{argmax}_{a \in \Gamma(x)} \left\{ r(x,a) + \beta \sum_{x' \in \mathsf{X}} v(x') P(x,a,x') \right\} \quad \text{for all } x \in \mathsf{X}$$

For each $v \in \mathbb{R}^{\mathsf{X}}$, a v-greedy policy exists

- just select a maximizer a_x^* from the nonempty set $\Gamma(x)$ at each x in X

Hence a v^*-greedy policy exists

Hence an optimal policy exists

Computing the Value of a Policy

How should we compute the value v_σ of a given policy σ?

We saw above that

$$v_\sigma = (I - \beta P_\sigma)^{-1} r_\sigma$$

- Computationally helpful when X is small
- But problematic for large dynamic programs

If, say, X has 10^6 elements, then $I - \beta P_\sigma$ is $10^6 \times 10^6$

Matrices of this size are difficult invert—or even store in memory

Another way to compute v_σ: use the **policy operator**

$$(T_\sigma v)(x) = r(x, \sigma(x)) + \beta \sum_{x' \in \mathsf{X}} v(x') P(x, \sigma(x), x')$$

- Defined at all $v \in \mathbb{R}^{\mathsf{X}}$
- Analogous to T_σ for the optimal stopping problem

In vector notation, we can write

$$T_\sigma v = r_\sigma + \beta P_\sigma v$$

- T_σ is order-preserving on $\mathbb{R}^{\mathsf{X}}$ — why?

Ex. Show that T_σ is a contraction of modulus β on $\mathbb{R}^X$

For any v, w in $\mathbb{R}^X$ we have

$$\begin{aligned}
|T_\sigma v - T_\sigma w| &= \beta\, |P_\sigma v - P_\sigma w| \\
&= \beta\, |P_\sigma (v - w)| \\
&\leqslant \beta\, P_\sigma\, |v - w| \\
&\leqslant \beta\, P_\sigma\, \|v - w\|_\infty \mathbb{1} \\
&= \beta\, \|v - w\|_\infty \mathbb{1}
\end{aligned}$$

Now use $|a| \leqslant |b|$ implies $\|a\|_\infty \leqslant \|b\|_\infty$

Ex. Show that T_σ is a contraction of modulus β on $\mathbb{R}^{\mathsf{X}}$

For any v, w in $\mathbb{R}^{\mathsf{X}}$ we have

$$
\begin{aligned}
|T_\sigma v - T_\sigma w| &= \beta \, |P_\sigma v - P_\sigma w| \\
&= \beta \, |P_\sigma (v - w)| \\
&\leqslant \beta \, P_\sigma \, |v - w| \\
&\leqslant \beta \, P_\sigma \, \|v - w\|_\infty \mathbb{1} \\
&= \beta \, \|v - w\|_\infty \mathbb{1}
\end{aligned}
$$

Now use $|a| \leqslant |b|$ implies $\|a\|_\infty \leqslant \|b\|_\infty$

Ex. Show that v_σ is the unique fixed point of T_σ in $\mathbb{R}^\mathsf{X}$

Proof: If $v = T_\sigma v$, then

$$v = r_\sigma + \beta P_\sigma v$$

Since $\beta < 1$, we then have

$$v = (I - \beta P_\sigma)^{-1} r_\sigma$$

Hence $v = v_\sigma$

Ex. Show that v_σ is the unique fixed point of T_σ in $\mathbb{R}^{\mathsf{X}}$

Proof: If $v = T_\sigma v$, then

$$v = r_\sigma + \beta P_\sigma v$$

Since $\beta < 1$, we then have

$$v = (I - \beta P_\sigma)^{-1} r_\sigma$$

Hence $v = v_\sigma$

Algorithms

Previously we used value function iteration (VFI) to solve optimal stopping problems

Here we

1. present a generalization suitable for aribtrary MDPs
2. introduce two other important methods

The two other methods are called

1. Howard policy iteration (HPI) and
2. Optimistic policy iteration (OPI)

Algorithm 2: VFI for MDPs

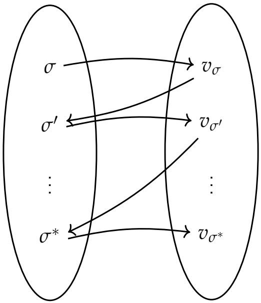
input $v_0 \in \mathbb{R}^{\mathsf{X}}$, an initial guess of v^*
input τ, a tolerance level for error
$\varepsilon \leftarrow \tau + 1$
$k \leftarrow 0$
while $\varepsilon > \tau$ **do**
  for $x \in \mathsf{X}$ **do**
    $v_{k+1}(x) \leftarrow (Tv_k)(x)$
  end
  $\varepsilon \leftarrow \|v_k - v_{k+1}\|_\infty$
  $k \leftarrow k + 1$
end
Compute a v_k-greedy policy σ
return σ

VFI is

- robust
- easy to implement and
- works relatively well in high dimensions after certain modifications

However, we can often find faster methods with a bit of effort

Howard Policy Iteration



Iterates between computing the value of a given policy and computing the greedy policy associated with that value

Algorithm 3: Howard policy iteration for MDPs

input $\sigma_0 \in \Sigma$, an initial guess of σ^*

$k \leftarrow 0$

$\varepsilon \leftarrow 1$

while $\varepsilon > 0$ **do**

  $v_k \leftarrow$ the σ_k-value function $(I - \beta P_{\sigma_k})^{-1} r_{\sigma_k}$

  $\sigma_{k+1} \leftarrow$ a v_k greedy policy

  $\varepsilon \leftarrow \|\sigma_k - \sigma_{k+1}\|_\infty$

  $k \leftarrow k + 1$

end

return σ_k

Advantages:

1. in a finite state setting, the algorithm always converges to the exact optimal policy in a finite number of steps
2. the rate of convergence is faster that VFI

We prove these facts in a more general setting when we discuss RDPs

Optimistic Policy Iteration

OPI borrows from both value function iteration and Howard policy iteration

The same as Howard policy iteration (HPI) except that

- HPI takes σ and obtains v_σ
- OPI takes σ and iterates m times with T_σ

Recall that $T_\sigma^m \to v_\sigma$ as $m \to \infty$

Hence OPI replaces v_σ with an approximation

Algorithm 4: Optimistic policy iteration for MDPs

input $v_0 \in \mathbb{R}^{\mathsf{X}}$, an initial guess of v^*
input τ, a tolerance level for error
input $m \in \mathbb{N}$, a step size
$k \leftarrow 0$
$\varepsilon \leftarrow \tau + 1$
while $\varepsilon > \tau$ **do**
    $\sigma_k \leftarrow$ a v_k-greedy policy
    $v_{k+1} \leftarrow T_{\sigma_k}^m v_k$
    $\varepsilon \leftarrow \|v_k - v_{k+1}\|_\infty$
    $k \leftarrow k + 1$
end
return σ_k

Regarding m,

- If $m = \infty$, OPI is identical to HPI
- If $m = 1$, OPI is identical to VFI

Usually, an intermediate value of m is better than both

- We investigate this in the applications below

The sequence $(\sigma_k)_{k \geqslant 1}$ always converges to an optimal policy

Application: Optimal Inventories

Previously we analyzed S-s inventory dynamics

- our aim was to understand Markov chains

But are such dynamics realistic?

We now investigate whether S-s behavior arises naturally in optimizing model

- firm chooses its inventory path to maximize firm value

We assume for now that the firm only sells one product

Given a demand process $(D_t)_{t \geqslant 0}$, inventory $(X_t)_{t \geqslant 0}$ obeys

$$X_{t+1} = m(X_t - D_{t+1}) + A_t$$

where

- $m(y) := y \vee 0$
- A_t is units of stock ordered this period
- The firm can store at most K items at one time

The state space is $\mathsf{X} := \{0, \ldots, K\}$

We assume $(D_t) \stackrel{\text{IID}}{\sim} \varphi \in \mathcal{D}(\mathbb{Z}+)$

Profits are given by

$$\pi_t := X_t \wedge D_{t+1} - cA_t - \kappa \mathbb{1}\{A_t > 0\}$$

- Orders in excess of inventory are lost
- c is unit product cost (and unit sales prices $= 1$)
- κ is a fixed cost of ordering inventory

With $\beta := 1/(1+r)$ and $r > 0$, the value of the firm is

$$V_0 = \mathbb{E} \sum_{t \geqslant 0} \beta^t \pi_t$$

Managers of the firm try to maximize shareholder value

Expected current profit is

$$r(x, a) := \sum_{d \geqslant 0} (x \wedge d) \varphi(d) - ca - \kappa \mathbb{1}\{a > 0\}$$

The set of feasible order sizes at x is

$$\Gamma(x) := \{0, \dots, K - x\}$$

The Bellman equation is

$$v(x) = \max_{a \in \Gamma(x)} \left\{ r(x, a) + \beta \sum_{d \geqslant 0} v(m(x - d) + a) \varphi(d) \right\}$$

An MDP with state space X and action space A := X

- Γ, r and β are as given above
- The stochastic kernel is

$$P(x, a, x') := \mathbb{P}\{m(x - D) + a = x'\} \quad \text{when} \quad D \sim \varphi$$

Since the inventory model is an MDP, all optimality results apply

- the unique fixed point of the Bellman operator is v^*
- a policy σ^* is optimal if and only if it is v^*-greedy
- etc.

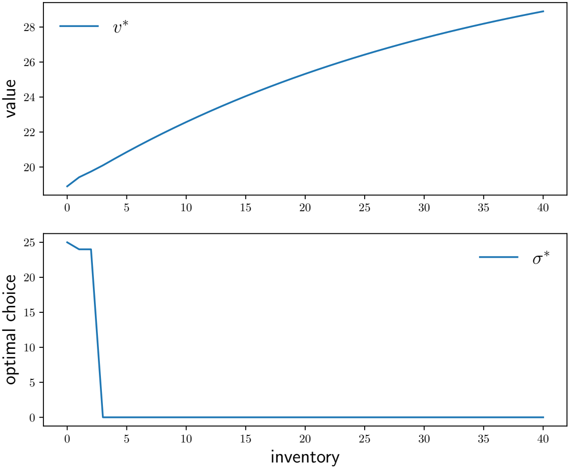
```julia
using Distributions, OffsetArrays
m(x) = max(x, 0)  # Convenience function

function create_inventory_model(; β=0.98,     # discount factor
                                  K=40,       # maximum inventory
                                  c=0.2, κ=2, # cost paramters
                                  p=0.6)      # demand parameter
    ϕ(d) = (1 - p)^d * p  # demand pdf
    return (; β, K, c, κ, p, ϕ)
end

"The function B(x, a, v) = r(x, a) + β Σ_x′ v(x′) P(x, a, x′)."
function B(x, a, v, model; d_max=100)
    (; β, K, c, κ, p, ϕ) = model
    reward = sum(min(x, d)*ϕ(d) for d in 0:d_max) - c * a - κ * (a > 0)
    continuation_value = β * sum(v[m(x - d) + a] * ϕ(d) for d in 0:d_max)
    return reward + continuation_value
end
```

```julia
"The Bellman operator."
function T(v, model)
    (; β, K, c, κ, p, ϕ) = model
    new_v = similar(v)
    for x in 0:K
        Γx = 0:(K - x)
        new_v[x], _ = findmax(B(x, a, v, model) for a in Γx)
    end
    return new_v
end

"Get a v-greedy policy.  Returns a zero-based array."
function get_greedy(v, model)
    (; β, K, c, κ, p, ϕ) = model
    σ_star = OffsetArray(zeros(Int32, K+1), 0:K)
    for x in 0:K
        Γx = 0:(K - x)
        _, a_idx = findmax(B(x, a, v, model) for a in Γx)
        σ_star[x] = Γx[a_idx]
    end
    return σ_star
end
```



v^*
value
28
26
24
22
20
0
5
10
15
20
25
30
35
40
σ^*
optimal choice
25
20
15
10
5
0
inventory

Ex. Try to replicate these plots

- Use the code given above (or at least the same parameters)
- Use value function iteration

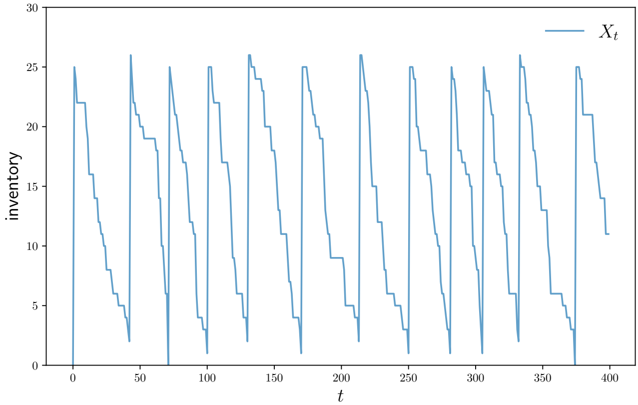


Figure: Optimal inventory dynamics

Optimal Savings with Labor Income

Wealth evolves according to

$$W_{t+1} = RW_t + Y_t - C_t \qquad (t = 0, 1, \ldots)$$

- (W_t) takes values in finite set $\mathsf{W} \subset \mathbb{R}_+$
- (Y_t) is Q-Markov chain on finite set Y
- $C_t, W_t \geqslant 0$

The household maximizes

$$\mathbb{E} \sum_{t \geqslant 0} \beta^t u(C_t)$$

The model is an MDP with state space $\mathsf{X} := \mathsf{W} \times \mathsf{Y}$

- The feasible correspondence is

$$\Gamma(w, y) = \{s \in \mathsf{W} : s \leqslant Rw + y\}$$

- The current reward is

$$r(w, y, s) = u(Rw + y - s)$$

- The stochastic kernel is

$$P((w, y), s, (w', y')) = \mathbb{1}\{w' = s\} Q(y, y')$$

Hence all MDP optimality results apply

Bellman operator:

$$(Tv)(w,y) =$$

$$\max_{w' \in \Gamma(w,y)} \left\{ u(Rw + y - w') + \beta \sum_{y' \in \mathsf{Y}} v(w', y') Q(y, y') \right\}$$

The policy operator for given $\sigma \in \Sigma$ is

$$(T_\sigma \, v)(w,y) =$$

$$u(Rw + y - \sigma(w,y)) + \beta \sum_{y' \in \mathsf{Y}} v(\sigma(w,y), y') Q(y, y')$$

How to solve $v_\sigma = (I - \beta P_\sigma)^{-1} r_\sigma$?

We set

$$P_\sigma((w,y),(w',y')) := \mathbb{1}\{\sigma(w,y) = w'\} Q(y,y')$$

and

$$r_\sigma(w,y) := u(Rw + y - \sigma(w,y))$$

How to use matrix algebra routines?

Set up a bijection $(i,j) \leftrightarrow m$ where

$$x_m = (w_i, y_j)$$

```julia
using QuantEcon, LinearAlgebra, IterTools

function create_savings_model(; R=1.01, β=0.99, γ=2.5,
                                w_min=0.01, w_max=5.0, w_size=200,
                                ρ=0.9, ν=0.1, y_size=5)
    w_grid = LinRange(w_min, w_max, w_size)
    mc = tauchen(y_size, ρ, ν)
    y_grid, Q = exp.(mc.state_values), mc.p
    return (; β, R, γ, w_grid, y_grid, Q)
end

"B(w, y, w′) = u(R*w + y - w′) + β Σ_y′ v(w′, y′) Q(y, y′)."
function B(i, j, k, v, model)
    (; β, R, γ, w_grid, y_grid, Q) = model
    w, y, w′ = w_grid[i], y_grid[j], w_grid[k]
    u(c) = c^(1-γ) / (1-γ)
    c = R*w + y - w′
    @views value = c > 0 ? u(c) + β * dot(v[k, :], Q[j, :]) : -Inf
    return value
end
```

```julia
"The Bellman operator."
function T(v, model)
    w_idx, y_idx = (eachindex(g) for g in (model.w_grid, model.y_grid))
    v_new = similar(v)
    for (i, j) in product(w_idx, y_idx)
        v_new[i, j] = maximum(B(i, j, k, v, model) for k in w_idx)
    end
    return v_new
end

"The policy operator."
function T_σ(v, σ, model)
    w_idx, y_idx = (eachindex(g) for g in (model.w_grid, model.y_grid))
    v_new = similar(v)
    for (i, j) in product(w_idx, y_idx)
        v_new[i, j] = B(i, j, σ[i, j], v, model)
    end
    return v_new
end
```

```julia
include("finite_opt_saving_0.jl")

"Compute a v-greedy policy."
function get_greedy(v, model)
    w_idx, y_idx = (eachindex(g) for g in (model.w_grid, model.y_grid))
    σ = Matrix{Int32}(undef, length(w_idx), length(y_idx))
    for (i, j) in product(w_idx, y_idx)
        _, σ[i, j] = findmax(B(i, j, k, v, model) for k in w_idx)
    end
    return σ
end
```

```julia
"Get the value v_σ of policy σ."
function get_value(σ, model)
    # Unpack and set up
    (; β, R, γ, w_grid, y_grid, Q) = model
    wn, yn = length(w_grid), length(y_grid)
    n = wn * yn
    u(c) = c^(1-γ) / (1-γ)
    # Function to extract (i, j) from m = i + (j-1)*wn"
    single_to_multi(m) = (m-1)%wn + 1, div(m-1, wn) + 1
    # Allocate and create single index versions of P_σ and r_σ
    P_σ = zeros(n, n)
    r_σ = zeros(n)
    for m in 1:n
        i, j = single_to_multi(m)
        r_σ[m] = u(R * w_grid[i] + y_grid[j] - w_grid[σ[i, j]])
        for m′ in 1:n
            i′, j′ = single_to_multi(m′)
            if i′ == σ[i, j]
                P_σ[m, m′] = Q[j, j′]
            end
        end
    end
    # Solve for the value of σ
    v_σ = (I - β * P_σ) \ r_σ
    # Return as multi-index array
    return reshape(v_σ, wn, yn)
end
```

```julia
"Value function iteration routine."
function value_iteration(model, tol=1e-5)
    vz = zeros(length(model.w_grid), length(model.y_grid))
    v_star = successive_approx(v -> T(v, model), vz, tolerance=tol)
    return get_greedy(v_star, model)
end

"Howard policy iteration routine."
function policy_iteration(model)
    wn, yn = length(model.w_grid), length(model.y_grid)
    σ = ones(Int32, wn, yn)
    i, error = 0, 1.0
    while error > 0
        v_σ = get_value(σ, model)
        σ_new = get_greedy(v_σ, model)
        error = maximum(abs.(σ_new - σ))
        σ = σ_new
        i = i + 1
        println("Concluded loop $i with error $error.")
    end
    return σ
end
```

```julia
"Optimistic policy iteration routine."
function optimistic_policy_iteration(model; tolerance=1e-5, m=100)
    v = zeros(length(model.w_grid), length(model.y_grid))
    error = tolerance + 1
    while error > tolerance
        last_v = v
        σ = get_greedy(v, model)
        for i in 1:m
            v = T_σ(v, σ, model)
        end
        error = maximum(abs.(v - last_v))
    end
    return get_greedy(v, model)
end
```

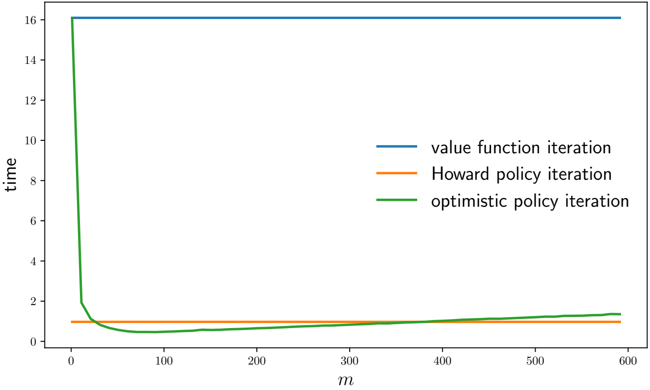


Figure: Timings for alternative algorithms

Optimal Savings: Transient vs Persistent Shocks

Keep all the same except the process for labor income

Now

$$Y_t = Z_t + \varepsilon_t \qquad (t \geqslant 0)$$

where

- $(Z_t)_{t \geqslant 0}$ is Q-Markov on Z
- $(\varepsilon_t)_{t \geqslant 0}$ is IID on E with distribution φ

Questions

- how does the household respond to persistent shocks?
- how does the household respond to transient shocks?
- how does this depend on wealth?

The model is an MDP with state

$$x := (w, z, \varepsilon) \in \mathsf{X} := \mathsf{W} \times \mathsf{Z} \times \mathsf{E}$$

- The feasible correspondence is

$$\Gamma(w, z, \varepsilon) = \{s \in \mathsf{W} : s \leqslant Rw + z + \varepsilon\}$$

- The current reward is $r(w, z, \varepsilon, s) = u(Rw + z + \varepsilon - s)$
- The stochastic kernel is

$$P((w, z, \varepsilon), s, (w', z', \varepsilon')) = \mathbb{1}\{w' = s\} Q(z, z') \varphi(\varepsilon')$$

Hence all of the MDP optimality results apply

Bellman operator:

$$(Tv)(w,z,\varepsilon) =$$

$$\max_{w' \in \Gamma(w,z,\varepsilon)} \left\{ u(Rw + z + \varepsilon - w') + \beta \sum_{z',\, \varepsilon'} v(w', z', \varepsilon') Q(z, z') \varphi(\varepsilon') \right\}$$

The policy operator for given $\sigma \in \Sigma$ is

$$(T_\sigma v)(w,z,\varepsilon) =$$

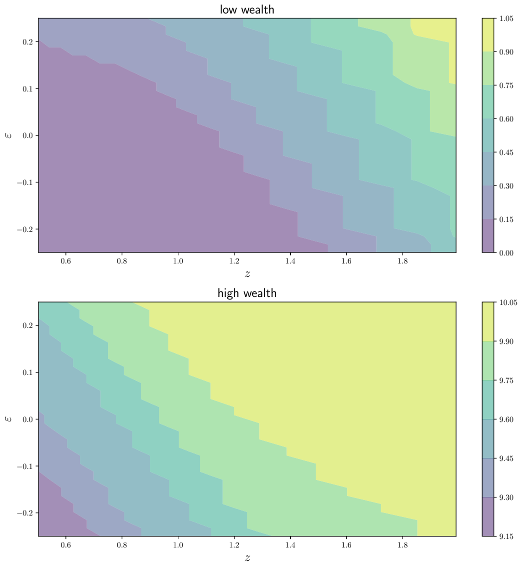
$$u(Rw + z + \varepsilon - \sigma(w,z,\varepsilon)) + \beta \sum_{z',\, \varepsilon'} v(\sigma(w,z,\varepsilon), z', \varepsilon') Q(z, z') \varphi(\varepsilon')$$

Let's look at optimal savings as a function of z and ε

We plot $(z, \varepsilon) \mapsto \sigma(w, z, \varepsilon)$

Fix w at

- $\min \mathsf{W}$
- $\max \mathsf{W}$


low wealth
0.2
0.1
0.0
−0.1
−0.2
ε
0.6
0.8
1.0
1.2
1.4
1.6
1.8
z
1.05
0.90
0.75
0.60
0.45
0.30
0.15
0.00
high wealth
10.05
9.90
9.75
9.60
9.45
9.30
9.15

Investment with Adjustment Costs

A monopolist faces an inverse demand function of the form

$$P_t = a_0 - a_1 Y_t + Z_t,$$

where

- a_0, a_1 are positive parameters
- Y_t is output
- P_t is price and
- the demand shock Z_t follows

$$Z_{t+1} = \rho Z_t + \sigma \eta_{t+1}, \qquad \{\eta_t\} \stackrel{\text{IID}}{\sim} N(0,1).$$

Current profits are given by

$$\pi_t := P_t Y_t - cY_t - \gamma(Y_{t+1} - Y_t)^2$$

- $\gamma(Y_{t+1} - Y_t)^2$ represents adjustment costs
- rapid changes to capacity are expensive when $\gamma > 0$

Objective: maximize value

$$\mathbb{E} \sum_{t=0}^{\infty} \beta^t \pi_t$$

- $\beta = 1/(1+r)$, where $r > 0$ is a fixed interest rate

Building intuition: What would happen if $\gamma = 0$?

- No intertemporal trade-off
- maximize current profit each period

Solve

$$\max_{Y_t} \{P_t Y_t - cY_t\} = \max_{Y_t} \{(a_0 - a_1 Y_t + Z_t) Y_t - cY_t\}$$

Ex. Show that the maximizer is

$$\bar{Y}_t := \frac{a_0 - c + Z_t}{2a_1}$$

On the other hand, if γ is very large then $(Y_t)_{t \geqslant 0}$ should be almost constant

Thus, we expect the following:

- If $\gamma \approx 0$, then Y_t will track $\bar{Y}_t$ closely
- If γ is large, then $(Y_t)_{t \geqslant 0}$ will be smoother than $(\bar{Y}_t)_{t \geqslant 0}$

In short,

$$\text{more adjustment costs} \implies \text{smoother time path for } (Y_t)_{t \geqslant 0}$$

Implementation as an MDP

Let $\mathsf{Y} \subset \mathbb{R}_+$ be a grid containing output values

We discretize (Z_t) using Tauchen's method

- Now (Z_t) is Q-Markov on finite set $\mathsf{Z} \subset \mathbb{R}$

The state space $\mathsf{X} := \mathsf{Y} \times \mathsf{Z}$

The action space is Y

The feasible correspondence is $\Gamma(x) = \mathsf{Y}$ for all x

- choice of output is not restricted by the state

The set Σ is all $\sigma \colon \mathsf{Y} \times \mathsf{Z} \to \mathsf{Y}$

The current reward function is current profits:

$$r(y, z, y') = (a_0 - a_1 y + z - c)y - \gamma(y' - y)^2$$

The stochastic kernel is

$$P((y, z), y', (y', z')) = \mathbb{1}\{y = y'\} Q(z, z')$$

Now the problem defines an MDP

- all of the optimality theory for MDPs applies

The Bellman operator for this problem is

$$(Tv)(y,z) = \max_{y' \in \mathbb{R}} \left\{ r(y,z,y') + \beta \sum_{z' \in \mathsf{Z}} v(y',z') Q(z,z') \right\}$$

Given $\sigma \in \Sigma$, we can express the policy operator as

$$(T_\sigma v)(y,z) = r(y,z,\sigma(y,z)) + \beta \sum_{z' \in \mathsf{Z}} v(\sigma(y,z),z') Q(z,z')$$

We know that both are

- order-preserving self-maps on $\mathbb{R}^{\mathsf{X}}$
- contraction maps on $\mathbb{R}^{\mathsf{X}}$

A v-greedy policy is a $\sigma \in \Sigma$ that obeys

$$\sigma(y,z) = \operatorname*{argmax}_{y' \in \mathsf{Y}} \left\{ r(y,z,y') + \beta \sum_{z' \in \mathsf{Z}} v(y',z') Q(z,z') \right\}$$

By our results for MDPs

- v^*-greedy policies = optimal policies
- optimistic policy iteration and VFI converge

Implications for output can be studied by

1. generating a Q-Markov chain $(Z_t)_{t=1}^T$
2. simulating optimal output via $Y_{t+1} = \sigma^*(Y_t, Z_t)$

```julia
using QuantEcon, LinearAlgebra, IterTools
include("s_approx.jl")

function create_investment_model(;
        r=0.04,                              # Interest rate
        a_0=10.0, a_1=1.0,                   # Demand parameters
        γ=25.0, c=1.0,                       # Adjustment and unit cost
        y_min=0.0, y_max=20.0, y_size=100,   # Grid for output
        ρ=0.9, ν=1.0,                        # AR(1) parameters
        z_size=25)                           # Grid size for shock
    β = 1/(1+r)
    y_grid = LinRange(y_min, y_max, y_size)
    mc = tauchen(y_size, ρ, ν)
    z_grid, Q = mc.state_values, mc.p
    return (; β, a_0, a_1, γ, c, y_grid, z_grid, Q)
```

```julia
"""
The aggregator B is given by

    B(y, z, y′) = r(y, z, y′) + β Σ_z′ v(y′, z′) Q(z, z′)."

where

    r(y, z, y′) := (a_0 - a_1 * y + z - c) y - γ * (y′ - y)^2

"""
function B(i, j, k, v, model)
    (; β, a_0, a_1, γ, c, y_grid, z_grid, Q) = model
    y, z, y′ = y_grid[i], z_grid[j], y_grid[k]
    r = (a_0 - a_1 * y + z - c) * y - γ * (y′ - y)^2
    return @views r + β * dot(v[k, :], Q[j, :])
end
```

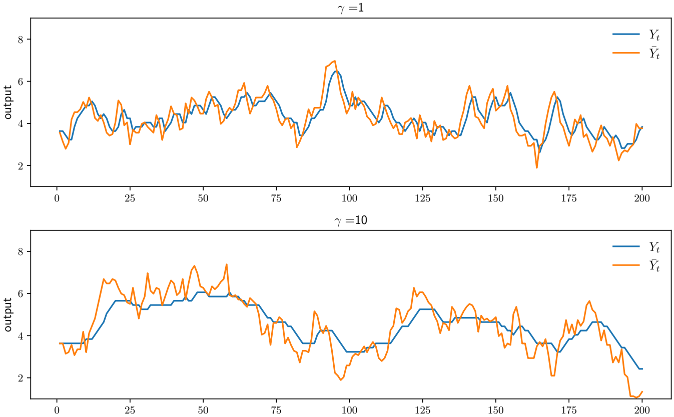
```julia
"The policy operator."
function T_σ(v, σ, model)
    y_idx, z_idx = (eachindex(g) for g in (model.y_grid, model.z_grid))
    v_new = similar(v)
    for (i, j) in product(y_idx, z_idx)
        v_new[i, j] = B(i, j, σ[i, j], v, model)
    end
    return v_new
end

"The Bellman operator."
function T(v, model)
    y_idx, z_idx = (eachindex(g) for g in (model.y_grid, model.z_grid))
    v_new = similar(v)
    for (i, j) in product(y_idx, z_idx)
        v_new[i, j] = maximum(B(i, j, k, v, model) for k in y_idx)
    end
    return v_new
end
```

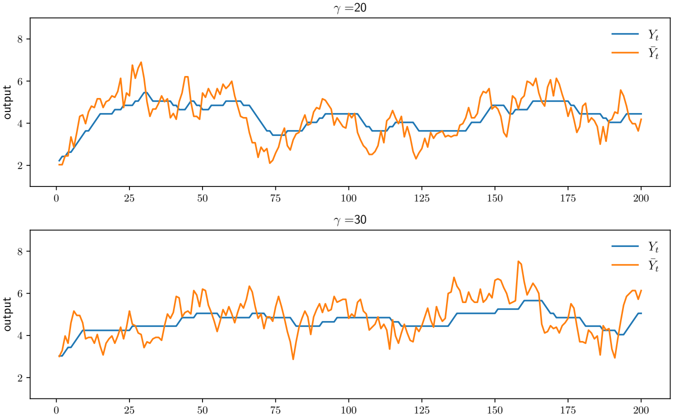
```julia
"Compute a v-greedy policy."
function get_greedy(v, model)
    y_idx, z_idx = (eachindex(g) for g in (model.y_grid, model.z_grid))
    σ = Matrix{Int32}(undef, length(y_idx), length(z_idx))
    for (i, j) in product(y_idx, z_idx)
        _, σ[i, j] = findmax(B(i, j, k, v, model) for k in y_idx)
    end
    return σ
end

"Value function iteration routine."
function value_iteration(model; tol=1e-5)
    vz = zeros(length(model.y_grid), length(model.z_grid))
    v_star = successive_approx(v -> T(v, model), vz, tolerance=tol)
    return get_greedy(v_star, model)
end
```

```julia
"Optimistic policy iteration routine."
function optimistic_policy_iteration(model; tol=1e-5, m=100)
    v = zeros(length(model.y_grid), length(model.z_grid))
    error = tol + 1
    while error > tol
        last_v = v
        σ = get_greedy(v, model)
        for i in 1:m
            v = T_σ(v, σ, model)
        end
        error = maximum(abs.(v - last_v))
    end
    return get_greedy(v, model)
end
```



γ =1
output
Y_t
$\bar{Y}_t$
8
6
4
2
0
25
50
75
100
125
150
175
200
γ =10
output
Y_t
$\bar{Y}_t$
8
6
4
2
0
25
50
75
100
125
150
175
200



γ =20
output
Y_t
$\bar{Y}_t$
γ =30
output
Y_t
$\bar{Y}_t$

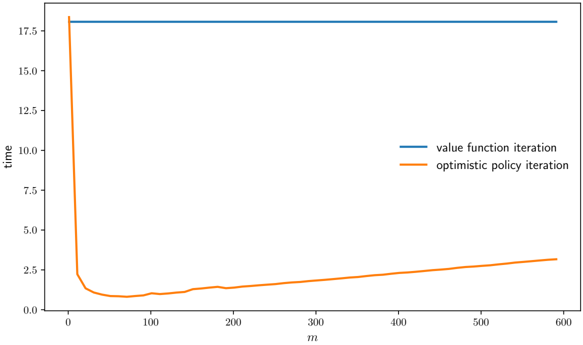


Figure: Timings for alternative algorithms, investment model

Notes on timing

- horizonal axis shows $m =$ step parameter in OPI
- vertical axis shows time in seconds
- Result for HPI not shown because time is 12x larger than VFI

Why is VFI faster than HPI here?

HPI tends to be strong when $\beta \approx 1$

- VFI convergence is linear in β, HPI convergence is quadratic

Here β is relatively small, so VFI beats HPI

Main messages

- OPI dominates both VFI and HPI for almost all values of m
- At $m = 60$, OPI is
    - 20 times faster than VFI
    - 240 times faster than HPI

Also note that OPI is easier to implement than HPI

- no need to map to single indices

Fixed Costs in Hiring and Firing

Consider a firm that maximizes expected present value

Future profits are discounted at rate

$$\beta = \frac{1}{1+r} \qquad r > 0$$

The only production input is labor

Hiring and firing involves fixed costs

Letting ℓ_t be employment, current profits are

$$\pi_t = pZ_t\ell_t^\alpha - w\ell_t - \kappa\mathbb{1}\{\ell_{t+1} \neq \ell_t\}$$

- p is the output price
- w is the wage rate
- α is a production parameter
- productivity $(Z_t)_{t \geqslant 0}$ is Q-Markov on Z and
- κ is a fixed cost of hiring and firing

Let $\mathsf{L} \subset \mathbb{R}_+$ be a finite grid for labor stock

The model is an MDP with state space $\mathsf{L} \times \mathsf{Z}$ and action space L

The feasible correspondence is

$$\Gamma(\ell, z) = \mathsf{L}$$

The reward function is

$$r(\ell, z, \ell') := pz\ell^{\alpha} - w\ell_t - \kappa \mathbb{1}\{\ell' \neq \ell\}$$

The stochastic kernel is

$$P((\ell, z), \ell', (\ell', z')) = \mathbb{1}\{\ell = \ell'\} Q(z, z')$$

Bellman operator:

$$(Tv)(\ell, z) = \max_{\ell' \in \Gamma(\ell, z)} \left\{ r(\ell, z, \ell') + \beta \sum_{z' \in \mathsf{Y}} v(\ell', z') Q(z, z') \right\}$$

The policy operator for given $\sigma \in \Sigma$ is

$$(T_\sigma v)(\ell, z) = r(\ell, z, \sigma(\ell, z)) + \beta \sum_{z' \in \mathsf{Y}} v(\sigma(\ell, z), z') Q(z, y')$$

A policy σ is v-greedy if

$$\sigma(\ell, z) \in \operatorname*{argmax}_{\ell' \in \Gamma(\ell, z)} \left\{ r(\ell, z, \ell') + \beta \sum_{z' \in \mathsf{Y}} v(\ell', z') Q(z, z') \right\}$$

```julia
using QuantEcon, LinearAlgebra, IterTools

function create_hiring_model(;
        r=0.04,                                 # Interest rate
        κ=1.0,                                  # Adjustment cost
        α=0.4,                                  # Production parameter
        p=1.0, w=1.0,                           # Price and wage
        l_min=0.0, l_max=30.0, l_size=100,      # Grid for labor
        ρ=0.9, ν=0.4, b=1.0,                    # AR(1) parameters
        z_size=100)                             # Grid size for shock
    β = 1/(1+r)
    l_grid = LinRange(l_min, l_max, l_size)
    mc = tauchen(z_size, ρ, ν, b, 6)
    z_grid, Q = mc.state_values, mc.p
    return (; β, κ, α, p, w, l_grid, z_grid, Q)
end
```

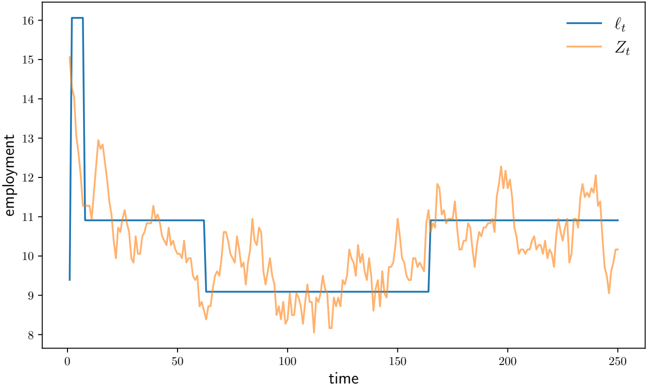


Figure: Fixed costs lead to jumps