Dynamic Programming

Chapter 0: Preface and Overview

Thomas J. Sargent and John Stachurski

2023

What is this book about?

These are the companion slides to Dynamic Programming (Vol 1)

Dynamic programming has a **vast** array of applications

- robotics
- artificial intelligence
- computational biology
- management science
- finance

Used daily to

- sequence DNA
- manage inventories
- control aircraft
- route shipping
- test products
- optimize database operations
- recommend products
- solve research problems
- etc., etc.

Example. Nvidia Hopper GPU DPX instruction sets are aimed specifically at accelerated dynamic programming

Within economics and finance, dynamic programming is applied to

- unemployment and search
- monetary policy and fiscal policy
- asset pricing and portfolio choice
- firm investment
- firm entry and exit
- wealth dynamics
- commodity pricing
- sovereign default
- economic geography
- international trade
- dynamic pricing, etc., etc., etc., etc.

What's covered in the book?

Standard dynamic programming topics, such as

- Markov decision processes
- Optimal stopping

Modern / advanced topics such as

- state-dependent discounting
- recursive preferences
- ambiguity / robust control
- transformations (Q-transforms, expected value functions, etc.)
- continuous time
- etc.

Common Symbols

$[m]$	the integers $\{1, \ldots, m\}$
$\mathbb{1}\{P\}$	equals 1 if statement P true, 0 otherwise
$\alpha := 1$	α is defined as equal to 1
$\alpha \equiv 1$	function α is everywhere equal to 1
$\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$	natural numbers, integers and real numbers
$\mathbb{C}$	complex numbers
$\mathbb{Z}_+$, $\mathbb{R}_+$, etc.	the nonnegative elements of $\mathbb{Z}$, $\mathbb{R}$, etc.
$\|x\|$ for $x \in \mathbb{R}$	absolute value of x
$\|\lambda\|$ for $\lambda \in \mathbb{C}$	modulus of λ
$a \vee b$	$\max\{a, b\}$
$a \wedge b$	$\min\{a, b\}$

Common Symbols

$\mathbb{1}$	a function (or vector) everywhere equal to 1
$\lvert B\rvert$	the cardinality of set B
$\mathbb{R}^n$	all n-tuples of real numbers
$x \leqslant y \;\; (x, y \in \mathbb{R}^n)$	$x_i \leqslant y_i$ for $i = 1, \ldots n$ (pointwise partial order)
$x \ll y \;\; (x, y \in \mathbb{R}^n)$	$x_i < y_i$ for $i = 1, \ldots n$
$\mathscr{D}(F)$	the set of distributions (or PMFs) on set F
$\mathbb{R}^{\mathsf{X}}$	all functions from X to $\mathbb{R}$
$\mathcal{L}(\mathbb{R}^{\mathsf{X}})$	the set of linear operators from $\mathbb{R}^{\mathsf{X}}$ to itself
$\mathcal{M}(\mathbb{R}^{\mathsf{X}})$	all $P \in \mathcal{L}(\mathbb{R}^{\mathsf{X}})$ with $P \geqslant 0$ and $P\mathbb{1} = \mathbb{1}$

Julia language: two minute introduction

We use Julia for embedded code because Julia is

- open source
- modern and well-designed
- efficient

Moreover Julia code can be close to underlying equations

- Makes it easy to write and debug

All source code can be found at

https://github.com/QuantEcon/book-dp1/

Note that

- Python code is also available
- Julia code is written for clarity, not speed

Julia Syntax

- Install from `https://julialang.org/` (if you wish)
- To import `Library`, write `using Library`
- `f(x) = 2x` defines the function $f(x) = 2x$
- `cos.(x)` applies `cos` to each elements of vector `x`
- `x.^2` squares each element of vector `x`
- loops / conditions very similar to Python
- data is pass by reference
- some nice multiple dispatch operations

```julia
# Defining functions, using conditions and loops

function f(x, y)                    # define a function
    if x < y                        # branch
        return sin(x + y)
    else
        return cos(x + y)
    end
end


function print_plurals(list_of_words)   # define a function
    for word in list_of_words           # loop
        println(word * "s")
    end
end
```

```julia
using LinearAlgebra          # import LinearAlgebra library

f(x) = 2x                    # simple function definition
f(x) = norm(x)               # norm defined in LinearAlgebra
g(x) = sum(x + x.^2)         # dot for pointwise operations
α, β = 2.0, -2.0             # unicode symbols

q(x) = sin(cos(x))           # another function
println(q(5))                # OK
x = rand(5)                  # build a random vector
println(q.(x))               # OK
println(q(x))                # error
```

The **composition** of $f\colon A \to B$ and $g\colon B \to C$ is $g \circ f\colon A \to C$ defined by

$$a \mapsto g(f(a))$$

Example. $f(x) = x \wedge 0$ and $g(x) = x \vee 0$ implies $g \circ f \equiv 0$

In Julia we can compose as follows

```
f(x) = min(x, 0)
g(x) = max(x, 0)
h = f ∘ g        # type \circ and then hit tab
```

Further information:

- QuantEcon resources: https://julia.quantecon.org
- Other resources: https://julialang.org/learning/

For advanced readers

For the rest of these chapter slides we discuss more advanced topics

- suitable for readers who already know some dynamic programming
- give more idea of what we cover in the book

Less advanced readers can safely skip to Chapter 1

Optimal consumption

Some readers will have studied an optimization problems such as

$$\max_{(C_t)_{t \geqslant 0}} \sum_{t \geqslant 0} \beta^t u(C_t)$$

subject to

$$W_{t+1} = R(W_t - C_t) \quad \text{and} \quad W_t, C_t \geqslant 0$$

- W_t is current wealth
- C_t is current consumption
- $u(C_t)$ is current utility (reward)
- β is a discount factor (time preference)

They will know how to set up the **Bellman operator**

$$(Tv)(w) = \max_{0 \leqslant c \leqslant w} \{u(c) + \beta v(R(w-c))\}$$

They will know that, under some conditions,

1. T is a contraction mapping
2. the unique fixed point of T is the value function v^*
3. v^* can be approximated via $v^* = \lim_{k \to \infty} T^k v$ for some v
4. optimal consumption at wealth w can be found by solving

$$c^* \in \operatorname*{argmax}_{0 \leqslant c \leqslant w} \{u(c) + \beta v^*(R(w-c))\}$$

But is this the best way?

Perhaps we should be using

- time iteration
- Howard policy iteration
- optimistic policy iteration

What are these algorithms?

Do they always converge?

Are they faster / more accurate?

Moreover, isn't our model too simplistic?

Possible extensions / modifications include

- labor income, work-leisure choice, multiple assets
- state-dependent discounting / preference shocks
- Epstein–Zin preferences
- ambiguity and ambiguity aversion
- robustness / risk-sensitivity / adversarial agents
- expected value formulation
- quantile preferences
- continuous time
- etc.

Example. What if discounting depends on the action (e.g., Uzawa preferences, as in, say, Schmitt-Grohé Uribe 2004)?

- β depends on consumption (and maybe labor/leisure)

$$(Tv)(w) = \max_{0 \leqslant c \leqslant w} \{u(c) + \beta(c)v(R(w-c))\}$$

- Is T still a contraction?
- Are all the previous optimality results still valid?
- Which algorithms converge?

Example. What if we face **state-dependent discounting** (as in, say, Krusell Smith 1998)?

- β now depends on an exogenous state process

$$(Tv)(w,z) = \max_{0 \leqslant c \leqslant w} \left\{ u(c) + \beta(z) \sum_{z'} v(R(w-c), z') Q(z, z') \right\}$$

- Is T still a contraction?
- Are all the previous optimality results still valid?
- Which algorithms converge?

Example. What about state-dependent discounting via **expected value functions** (as in the structural estimation literature), where

$$g(w, z, c) := \sum_{z'} v(R(w - c), z') Q(z, z')$$

with "Bellman operator"

$$(Hg)(w, z, c) =$$

$$\sum_{z'} \max_{0 \leqslant c' \leqslant R(w-c)} \left\{ u(c') + \beta(z') g(R(w - c), z', c') \right\} Q(z, z')$$

Does H have all the same properties as T?

What are the equivalent algorithms and do they converge?

And what happens if we introduce **Epstein–Zin preferences** (e.g., Bansal Yaron 2004, Basu Bundick 2017)?

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

$$\max_{0 \leqslant c \leqslant w} \left\{ c^\alpha + \beta(z) \left[\sum_{z'} v(R(w-c), z')^\gamma Q(z,z') \right]^{\alpha/\gamma} \right\}^{1/\alpha}$$

- Is T still a contraction?
- Are all the previous optimality results still valid?
- Which algorithms converge?

Or **risk-sensitive preferences** (as in, say, Hansen Sargent 2007)?

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

$$\max_{0 \leqslant c \leqslant w} \left\{ u(c) + \frac{\beta(z)}{\theta} \ln \left[\sum_{z'} \mathrm{e}^{\theta v(R(w-c), z')} Q(z, z') \right] \right\}$$

- Is T still a contraction?
- Are all the previous optimality results still valid?
- Which algorithms converge?

And what happens if we introduce

- quantile preferences?
- adversarial agents?
- ambiguity?

Is the Bellman operator a contraction?

Do the optimality properties hold?

Which algorithms converge?

How do we compute solutions?

We will address these questions by

1. covering the basic, foundational models
2. introducing state-dependent discounting
3. introducing recursive preferences
4. developing a general framework to handle all of the above
5. studying optimality and algorithms in the general framework
6. studying relationships between dynamic programs
7. switching to continuous time

A quick sketch of the main ideas:

Our general analysis uses **abstract Bellman equations** of the form

$$v(x) = \max_{a \in \Gamma(x)} B(x, a, v)$$

Includes all models discussed so far as special cases

Example. For the Epstein–Zin specification on slide 23, we use

$$B(x, a, v) = B((w, z), c, v) =$$

$$\left\{ c^\alpha + \beta(z) \left[\sum_{z'} v(R(w-c), z')^\gamma Q(z, z') \right]^{\alpha/\gamma} \right\}^{1/\alpha}$$

Behavior is determined via policy functions

A **feasible policy** is a map $\sigma \colon \mathsf{X} \to \mathsf{A}$ such that

$$\sigma(x) \in \Gamma(x) \text{ for all } x \in \mathsf{X}$$

- always respond to state x with action $x := \sigma(x)$

Let $\Sigma =$ the set of all feasible policies

Given $\sigma \in \Sigma$, suppose v_σ satisfies

$$v_\sigma(x) = B(x, \sigma(x), v_\sigma) \quad \text{for all } x \in \mathsf{X}$$

Interpretation:

- $v_\sigma(x) =$ **lifetime value** of policy σ given $X_0 = x$

Questions

- Is this interpretation reasonable?
- When is v_σ well defined / uniquely defined?

Suppose v_σ is uniquely defined for all $\sigma \in \Sigma$

Then we can introduce the **value function** via

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

$$= \text{max lifetime value from state } x$$

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

$$v_\sigma = v^*$$

Under what conditions do standard optimality results hold?

- does v^* satisfy $v^*(x) = \max_{a \in \Gamma(x)} B(x, a, v^*)$?
- does an optimal policy exist?
- are optimal policies characterized by Bellman's principle, where

$$\sigma(x) \in \operatorname*{argmax}_{a \in \Gamma(x)} B(x, a, v^*)$$

- under what conditions on B do the usual algorithms converge?
- what transformations can we apply to B in order to simplify analysis / computation?

We address these questions step by step

1. Provide general conditions
2. Provide more specialized conditions for special cases
   - contractive models
   - eventually contractive models
   - concave models / convex models
3. Connect these conditions to applications

We also discuss "completely abstract" DP models that can handle additional applications

- See Ch 9

Advantages

Working in this abstract setting

- simplifies proofs
- clarifies theory
- clarifies relationships between "similiar" dynamic programs

Notice

- no explicit dynamics
- no measure theory

Such details are pushed back to applications

- are the conditions we place on B satisfied?

Qualifications

Most of the current volume focuses on finite states and actions

This is not a bad thing because

- eliminates lots of qualifying statements
- minimizes functional analysis / measure theory
- covers all computable models

For general state spaces, see

1. the abstract theory in Ch 9 and
2. Vol II (forthcoming!)

Structure

The book starts with concrete problems

1. finite horizon search
2. infinite horizon search
3. Markov dynamics
4. optimal stopping
5. Markov decision processes
6. state-dependent Markov decision processes

Then we shift to abstract problems (Ch.s 7–9)