Dynamic Programming and Stochastic Control

Dr. Alex Leong

Department of Electrical Engineering (EIM-E)
Paderborn University, Germany
alex.leong@upb.de
1 Introduction
Introduction

What is dynamic programming (DP)?

- Method for solving multi-stage decision problems (Sequential decision making).
- There is often some randomness to what happens in future.
- Optimize set of decisions to achieve a good overall outcome.
- Richard Bellman popularized DP in the 1950s
Examples

1) **Inventory control**

- A store sells a product, e.g. Ice cream.
- Order supplies once a week.
- Sales during the week are “random”.
- How much supply should the store get to maximize *expected* profit over summer?
  - Order too little, can’t meet demand.
  - Order too much, storage/refrigeration cost.
Examples

2) Parts replacement e.g. bus engine.

- At the start of each month, decide whether the engine on a bus should be replaced, to maximize expected profit?
- If replace, profit = earnings - replacement cost - maintenance.
- If don’t replace, profit = earnings - maintenance.
- Earnings will decrease if engine breaks down.
- $P(\text{Breakdown})$ is age dependent.
Examples

3) Formula 1 engines, replace or not?

- 20 races, 4 engines (in 2017)
- Decide whether to replace engine at the start of each race, to maximize chance of winning championship.
Examples

4) Queueing (see Figure 1)

- Packets arrive at queues 1 and 2.
- If both queues transmit at the same time, have collision.
- If collision, retransmit at the next time with a certain probability.
- Choose retransmission probabilities to maximize throughput.

![Figure 1: Queueing](image)
5) LQR (Linear Quadratic Regulator)
Linear System: $x_{k+1} = Ax_k + Bu_k$ (Deterministic Problem)
- Assume knowledge of $x_k$ at time $k$ (Perfect state info)
- Choose sequence of $u_k$ to

$$\min_{u_0, u_1, \ldots, u_{N-1}} \sum_{k=0}^{N-1} (x_k^T Q x_k + u_k^T R u_k) + x_N^T Q x_N$$

- $N =$ number of stages $=$ horizon.
  - $N$ finite $\rightarrow$ finite horizon.
Examples

6) $x_{k+1} = Ax_k + Bu_k + w_k$
   - $w_k = \text{Random noise.}$
   - Assume $x_k$ known (Perfect state info)
   - Choose sequence of $u_k$ to

$$
\min_{u_0, u_1, \ldots, u_{N-1}} \mathbb{E} \left[ \sum_{k=0}^{N-1} (x_k^T Q x_k + u_k^T R u_k) + x_N^T Q x_N \right]
$$
Examples

7) LQG (Linear Quadratic Gaussian) Control

\[ x_{k+1} = Ax_k + Bu_k + w_k \]
\[ y_k = Cx_k + v_k \]

- \( v_k, w_k \) Gaussian noise.
- Case of imperfect state info.
- Based on measurements \( y_k \), choose \( u_k \) to

\[
\min_{u_0, u_1, \ldots, u_{N-1}} \mathbb{E} \left[ \sum_{k=0}^{N-1} (x_k^T Q x_k + u_k^T R u_k) + x_N^T Q x_N \right]
\]
8) Infinite horizon

\[
\min_{u_0, u_1, \ldots, u_{N-1}} \lim_{N \to \infty} \frac{1}{N} \mathbb{E} \left[ \sum_{k=0}^{N-1} (x_k^T Q x_k + u_k^T R u_k) + x_N^T Q x_N \right]
\]

Note: Here we divide by \( N \), otherwise summation often blows up.
Examples

9) Shortest paths (see Figure 2)

- Find shortest path from A to stage D (Deterministic Problem).
- Can solve using the Viterbi algorithm (1967)
- Can be regarded as a special case of (forward) DP.
- Applications:
  - decoding of convolutional codes (communications)
  - channel equalization (communications)
  - estimation of hidden Markov models (signal processing)

![Figure 2: Shortest paths problem](image)
Outline

2 The Dynamic Programming Principle and Dynamic Programming Algorithm
   • Basic Structure of Dynamic Programming Problem
   • Dynamic Programming Principle of Optimality
   • Dynamic Programming Algorithm
   • Shortest Path Problems
Basic structure of stochastic DP problem

Two ingredients, discrete time system and cost function

1. Discrete time system

\[ x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, \ldots, N - 1 \quad \text{(or } k = 1, 2, \ldots, N) \]

- \( k \) is time index.
- \( x_k \) is state at time \( k \), summarizes past information that is relevant for future optimization.
- \( u_k \) is control/decision/action at time \( k \), lies in a set \( U_k(x_k) \) which may depend on \( k \) and \( x_k \).
- \( w_k \) is random disturbance (noise), with a probability distribution \( P(.|k, x_k, u_k) \) which may depend on \( k, x_k, u_k \).
Basic structure of stochastic DP problem

\[ x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, \ldots, N - 1 \]

- \( N \) is horizon, or number of times control is applied.
- \( f_k \) is function that describes how system evolves over time.
- Examples
  - \( f_k = Ax_k + Bu_k + w_k \) (linear system)
  - \( f_k = x_k u_k + w_k \) (non-linear)
  - \( f_k = \cos x_k + w_k \sin u_k \) (non-linear)
Basic structure of stochastic DP problem

2. Cost function which is additive over time

\[ \mathbb{E} \left[ \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N) \right] \]

- Expectation is used because of random \( w_k \).
- \( g_k \) is function that represents cost at time \( k \).
- Examples
  - \( g_k = x_k + u_k \)
  - \( g_k = x_k^2 + C u_k^2 \), where \( C \) is a constant.
- \( g_N(x_N) \) is terminal cost.
Basic structure of stochastic DP problem

Objective: Minimize the cost function over the controls

\[ u_0 = \mu_0(x_0), \quad u_1 = \mu_1(x_1), \ldots, \quad u_{N-1} = \mu_{N-1}(x_{N-1}) \]

- Choice of \( u_k \) depends on \( x_k \).
- Optimization over policies: rules/functions \( \mu_k \) for generating \( u_k \) for every possible value of \( x_k \).
- Expected cost of policy \( \pi = (\mu_0, \mu_1, \ldots, \mu_{N-1}) \) starting at \( x_0 \) is

\[
J_\pi(x_0) = \mathbb{E} \left[ \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) + g_N(x_N) \right]
\]

- Optimal policy: \( \pi^* = \arg\min_\pi J_\pi(x_0) \)
- Optimal cost starting at \( x_0 \): \( J^*(x_0) = \min_\pi J_\pi(x_0) \)
Examples

1) Inventory example

\[ x_k = \text{amount of stock at time } k. \]
\[ u_k = \text{stock ordered at time } k. \]
\[ w_k = \text{demand at time } k, \text{ with some probability distribution e.g. uniform.} \]

System: \[ x_{k+1} = x_k + u_k - w_k \ (= f_k(x_k, u_k, w_k)) \]

- \( x_k \) can be negative with this model.
- Alternative model: \( x_{k+1} = \max(0, x_k + u_k - w_k) \).

Cost function at time \( k \): \[ g_k(x_k, u_k, w_k) = r(x_k) + Cu_k \]
- \( r(x_k) \) is penalty for holding excess stock.
- \( C \) is cost per item.
1) Inventory example (cont.)

Terminal cost: \( R(x_N) \) is penalty for having excess stock at the end.

Cost function: \( \mathbb{E} \left[ \sum_{k=0}^{N-1} (r(x_k) + C u_k) + R(x_N) \right] \)

- Amount \( u_k \) to order can depend on inventory level \( x_k \).
- Can have constraints on \( u_k \), e.g. \( x_k + u_k \leq \text{max. storage} \).
- Optimization over policies: Find the rule which tells you how much to order for every possible stock level \( x_k \).
Examples

2) Example 6 of previous section

System

\[ x_{k+1} = A x_k + B u_k + w_k \]

Cost function

\[ \mathbb{E} \left[ \sum_{k=0}^{N-1} \left( x_k^T Q x_k + u_k^T R u_k \right) + x_N^T Q x_N \right] \]

- Objective: Determine \( u_k = \mu_k(x_k) \), \( k = 0, 1, \ldots, N - 1 \), to minimize the cost function.
- Solution turns out to be \( u_k^* = L_k x_k \) for some matrices \( L_k \). (Derived in later lecture)
Examples

3) Shortest paths (see Figure 3)

Figure 3: Shortest path problem

\(x_k = \) which node we’re in at stage \(k\).

\(u_k = \) which path we take to get to stage \(k + 1\)

\(w_k = \) zero

Cost function = Sum of values along the paths we choose.
Open loop vs. Closed loop

Open loop: Controls \((u_0, u_1, \ldots, u_{N-1})\) chosen at beginning (time 0).
Closed loop: Policy \((\mu_0, \mu_1, \ldots, \mu_{N-1})\) chosen, where at time \(k\),
\[ \mu_k(x_k) = u_k \] can depend on \(x_k\).

- Can adapt to conditions.
- e.g. Inventory problem. If current stock level:
  - \(x_k\) high \(\rightarrow\) order less.
  - \(x_k\) low \(\rightarrow\) order more.

- Closed loop is always at least as good as open loop.
- For deterministic problems, open loop is as good as closed loop
  - can predict exactly the future states given initial state and sequence of controls.
- For stochastic problems, generally should use closed loop.
D.P. Principle of Optimality

Intuition

Consider the shortest path problem in Figure 4.

- Shortest path from A to F shown in red: A → C → D → F
- Shortest path from C to F: C → D → F.
  - Subpath of shortest path from A → F.
- Shortest path from D to F: D → F.
  - Subpath of shortest path from A → F.
Observation
Shortest path from A to F contains shortest paths from intermediate nodes to F.

Why?
- Suppose there is a shorter path from C to F which is not C→D→F.
- Then can construct a new path A→C→...→F (new shortest path) which is shorter than A→C→D→F
  ⇒ contradicts A→C→D→F being the shortest.
D.P. Principle of Optimality

Formal statement:

- Basic problem

\[
\min_{\pi} \mathbb{E} \left( \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) + g_N(x_N) \right)
\]

- Let \( \pi^* = \{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\} \) be the optimal policy. Consider the “tail subproblem”

\[
\min_{\mu_i, \mu_{i+1}, \ldots, \mu_{N-1}} \mathbb{E} \left( \sum_{k=i}^{N-1} g_k(x_k, \mu_k(x_k), w_k) + g_N(x_N) \right),
\]

where we are at state \( x_i \) at time \( i \) and we wish to minimize the “cost to go” from time \( i \) to time \( N \).

- D.P. Principle of optimality then says that \( \{\mu_i^*, \mu_{i+1}^*, \ldots, \mu_{N-1}^*\} \) is optimal for the tail subproblem.
D.P. Principle of Optimality

“Proof”:
If \( \{\bar{\mu}_0, \bar{\mu}_1, \ldots, \bar{\mu}_{N-1}\} \) is a better policy for tail subproblem, then
\( \{\mu^*_0, \mu^*_1, \ldots, \mu^*_{i-1}, \bar{\mu}_i, \ldots, \bar{\mu}_{N-1}\} \) would be a better policy for original problem
\( \Rightarrow \) contradiction of \( \{\mu^*_0, \mu^*_1, \ldots, \mu^*_{N-1}\} \) being optimal.

How can we make use of the D.P. principle?
Idea: Construct an optimal policy in stages.

- Solve tail subproblem involving last stage, to obtain \( \mu^*_{N-1} \)
- Solve tail subproblem involving last two stages, making use of \( \mu^*_{N-1} \),
  to obtain \( \mu^*_{N-2} \)
- Solve tail subproblem involving last three stages, making use of
  \( \mu^*_{N-2}, \mu^*_{N-1} \), to obtain \( \mu^*_{N-3} \)
  ...
- Solve tail subproblem involving last \( N \) stages, making use of
  \( \mu^*_1, \ldots, \mu^*_{N-1} \), to obtain \( \mu^*_0 \)
D.P. Algorithm

- Basic problem:

\[
\min_{\pi} E \left\{ \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) + g_N(x_N) \right\}
\]

- D.P. Algorithm: For each possible \( x_k \), compute:

\[
J_N(x_N) = g_N(x_N),
\]
\[
J_k(x_k) = \min_{u_k \in U_k(x_k)} E\{g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))\},
\]

for \( k = N - 1, N - 2, \ldots, 1, 0 \)

Theorem:

1. Optimal cost \( J^*(x_0) = J_0(x_0) \), where \( J_0(x_0) \) is quantity computed by D.P. algorithm.
2. Let \( \mu_k^*(.) \) be the function that generates the minimum \( u_k \) in the D.P. algorithm, i.e. \( \mu_k^*(x_k) = u_k^* \). Then \( \{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\} \) is the optimal policy to the basic problem.

Proof: See later
D.P. Algorithm

Comments:
- D.P. algorithm needs to be run for all possible states $x_k$.
- Solves all tail subproblems (don’t know which subproblem you need at the start).
- Can be computationally expensive if number of states/controls is large.
- Often done on computer.
- Suboptimal methods can reduce complexity.
Inventory Example

- $x_k =$ level of stock at time $k$.
- $u_k =$ amount ordered at time $k$.
- $w_k =$ demand at time $k$.
- $x_{k+1} = \max(0, x_k + u_k - w_k) = f_k(x_k, u_k, w_k)$, excess demand is lost.
- Storage constraint: $x_k + u_k \leq 2$
- Cost at time $k$
  \[
  \text{Cost} = \underbrace{\text{Purchasing cost}}_{\text{cost per item}=1 \text{euro}} + \underbrace{\text{storage cost}}_{(x_k + u_k - w_k)^2} = u_k + (x_k + u_k - w_k)^2 = g_k(x_k, u_k, w_k)
  \]
- Terminal cost $g_N(x_N) = 0$.
- Probability distribution of $w_k$:
  \[
  \mathbb{P}(w_k = 0) = 0.1, \mathbb{P}(w_k = 1) = 0.7, \mathbb{P}(w_k = 2) = 0.2
  \]
Inventory Example

Problem: Find the optimal policy for horizon \( N = 3 \), i.e.

\[
\min_{(\mu_0, \mu_1, \mu_2)} \mathbb{E} \left\{ \sum_{k=0}^{2} g_k(x_k, \mu_k(x_k), w_k) \right\}
\]

Apply D.P. algorithm:

\( J_3(x_3) = g_3(x_3) = 0 \)

\( J_k(x_k) = \min_{u_k \in U_k} \mathbb{E}\{u_k + (x_k + u_k - w_k)^2 + J_{k+1}(\max(0, x_k + u_k - w_k))\}, \quad k = 2, 1, 0 \)

Question: What values can \( x_k \) take?
Inventory Example

Period 2:
Compute $J_2(x_2)$ for all possible values of $x_2$

$$J_2(0) = \min_{u_2 \in \{0, 1, 2\}} \mathbb{E}\left\{ u_2 + (0 + u_2 - w_2)^2 + J_3(x_3) \right\}$$

$$= \min_{u_2 \in \{0, 1, 2\}} u_2 + \mathbb{E}\{(u_2 - w_2)^2\}$$

$$= \min_{u_2 \in \{0, 1, 2\}} u_2 + (u_2 - 0)^2 0.1 + (u_2 - 1)^2 0.7 + (u_2 - 2)^2 0.2$$

- If $u_2 = 0$: $u_2 + 0.1u_2^2 + 0.7(u_2 - 1)^2 + 0.2(u_2 - 2)^2 = 0.7 \times 1 + 0.2 \times 4 = 1.5$
- If $u_2 = 1$: $1 + 0.1 \times 1 + 0.7 \times 0 + 0.2 \times 1 = 1.3$
- If $u_2 = 2$: $2 + 0.1 \times 4 + 0.7 \times 1 + 0.2 \times 0 = 3.1$

$\Rightarrow J_2(0) = 1.3$ and $\mu_2^*(0) = 1$
Inventory Example

\[ J_2(1) = \min_{u_2 \in \{0,1\}} u_2 + (1 + u_2)^2 0.1 + (1 + u_2 - 1)^2 0.7 + (1 + u_2 - 2)^2 0.2 \]

- If \( u_2 = 0 \): 0.3 (check this!)
- If \( u_2 = 1 \): 2.1
  \[ \Rightarrow J_2(1) = 0.3 \text{ and } \mu^*_2(1) = 0 \]

\[ J_2(2) = \min_{u_2 \in \{0\}} \mathbb{E}\{u_2 + (2 + u_2 - w_2)^2\} = \cdots = 1.1 \]
\[ \Rightarrow J_2(2) = 1.1 \text{ and } \mu^*_2(2) = 0. \]
Inventory Example

Period 1:
Compute $J_1(x_1)$ for all possible values of $x_1$.

$$J_1(0) = \min_{u_1 \in \{0, 1, 2\}} \mathbb{E}\{u_1 + (u_1 - w_1)^2 + J_2(\max(0, 0 + u_1 - w_1))\}$$

$$= \min_{u_1 \in \{0, 1, 2\}} u_1 + (u_1^2 + J_2(\max(0, u_1)))0.1$$

$$+ ((u_1 - 1)^2 + J_2(\max(0, u_1 - 1)))0.7$$

$$+ ((u_1 - 2)^2 + J_2(\max(0, u_1 - 2)))0.2$$

- $u_1 = 0$: $J_2(0) \times 0.1 + (1 + J_2(0))0.7 + (4 + J_2(0))0.2 = 2.8$
  - from previous stage

- $u_1 = 1$: $1 + (1 + J_2(1))0.1 + J_2(0)0.7 + (1 + J_2(0))0.2 = 2.5$
  - from previous stage

- $u_1 = 2$: $2 + (4 + J_2(2))0.1 + (1 + J_2(1))0.7 + J_2(0)0.2 = 3.6$
  $$\Rightarrow J_1(0) = 2.5 \text{ and } \mu^*_1(0) = 1$$
Inventory Example

\[ J_1(1) = \min_{u_1 \in \{0, 1\}} \mathbb{E}\{u_1 + (1 + u_1 - w_1)^2 + J_2(\max(0, 1 + u_1 - w_1))\} \]

- \( u_1 = 0 \): 1.5 (check!)
- \( u_1 = 1 \): 2.68
  \[ \Rightarrow J_1(1) = 1.5, \text{ and } \mu_1^*(1) = 0 \]

\[ J_1(2) = 1.68, \mu_1^*(2) = 0 \text{ (check!) } \]

**Period 0:**
Compute \( J_0(x_0) \) for all possible \( x_0 \) (Tutorial problem)
Solution: \( J_0(0) = 3.7, J_0(1) = 2.1, J_0(2) = 2.818 \)
\( \mu_0^*(0) = 1, \mu_0^*(1) = 0, \mu_0^*(2) = 0 \)
**Example**: Scheduling problem (deterministic problem)

- Four operations need to be performed: A, B, C, D.
- B has to occur after A, D has to occur after C.
- Costs: \( c_{AB} = 2, c_{AC} = 3, c_{AD} = 4, c_{BC} = 3, c_{BD} = 1, c_{CA} = 4, c_{CB} = 4, c_{CD} = 6, c_{DA} = 3, c_{DB} = 3. \)
- Startup costs: \( S_A = 5, S_C = 3. \)

What is the optimal order?
Scheduling Example

Figure: Scheduling

Minimum cost to go in red
Scheduling Example

Use D.P. algorithm

- Let State = Set of operations already performed, see Figure “Scheduling”.
- No terminal costs for this problem.

Tail subproblems of length 1.
- Easy, only one choice at each state, e.g. if state [ACD], next operation has to be B.

Tail subproblems of length 2.
- State [AB], only one choice, next operation is C.
- State [AC], if next operation is B: cost = 4 + 1 = 5.
  State [AC], if next operation is D: cost = 6 + 3 = 9. ⇒ Choose B.
- State [CA], if next operation is B: cost = 2 + 1 = 3.
  State [CA], if next operation is D: cost = 4 + 3 = 7. ⇒ Choose B.
- State [CD], only one choice, next operation is A.
Scheduling Example

Tail subproblems of length 3.

- State $A$, if next operation is $B$: cost = $2 + 9 = 11$.
- State $A$, if next operation is $C$: cost = $3 + 5 = 8$. $\Rightarrow$ Choose C
- State $C$, if next operation is $A$: cost = $4 + 3 = 7$.
- State $C$, if next operation is $D$: cost = $6 + 5 = 11$. $\Rightarrow$ Choose A.

Original problem of length 4.

- If start with $A$: cost = $5 + 8 = 13$
- If start with $C$: cost = $3 + 7 = 10$ $\Rightarrow$ Choose C

Therefore, the optimal sequence = $\text{CABD}$, and the optimal cost = 10.
Proof that D.P. Algorithm gives Optimal Solution

Basic problem:

\[
\min_{\pi} \mathbb{E} \left\{ \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) + g_N(x_N) \right\}
\]

D.P. Algorithm: For each possible \( x_k \), compute:

\[
J_N(x_N) = g_N(x_N),
\]
\[
J_k(x_k) = \min_{u_k \in U_k(x_k)} \mathbb{E}\{g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))\},
\]

for \( k = N - 1, N - 2, \ldots, 1, 0 \)

Theorem:

1. Optimal cost \( J^*(x_0) = J_0(x_0) \), where \( J_0(x_0) \) is quantity computed by D.P. algorithm.

2. Let \( \mu_k^*(.) \) be the function that generates the minimum \( u_k \) in D.P. algorithm i.e \( \mu_k^*(x_k) = u_k^* \). Then \( \{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\} \) is the optimal policy to the basic problem.
Proof that D.P. Algorithm gives Optimal Solution

Notation:

Given policy $\pi = (\mu_0, \mu_1, \ldots, \mu_{N-1})$,
let $\pi^k = (\mu_k, \mu_{k+1}, \ldots, \mu_{N-1})$ = “tail policy”
and $J^*_k(x_k) = \min_{\pi^k} \mathbb{E}\{\sum_{i=k}^{N-1} g_i(x_i, \mu_i(x_i), w_i) + g_N(x_N)\}$ be the optimal cost for tail subproblem.

Let $J_k(x_k) =$ quantity computed by D.P algorithm.

- Want to show that $J^*_k(x_k) = J_k(x_k)$, for all $x_k, k$.
- Proof is by mathematical induction

Initial step ($k = N$):
- By definition of $J^*_k(x_k)$, $J^*_N(x_N) = g_N(x_N)$
- By definition of D.P algorithm $J_N(x_N) = g_N(x_N)$
  $\Rightarrow J^*_N(x_N) = J_N(x_N)$
Proof that D.P. Algorithm gives Optimal Solution

Induction step:

- Assume $J^*_l(x_l) = J_l(x_l)$ for $l = N, N - 1, ..., k + 1$
- Want to show that $J^*_k(x_k) = J_k(x_k)$
- From the definition of $J^*_k(x_k)$,

$$J^*_k(x_k) = \min_{\pi^k} \mathbb{E} \left\{ \sum_{i=k}^{N-1} g_i(x_i, \mu_i(x_i), w_i) + g_N(x_N) \right\}$$

$$= \min_{(\mu_k, \pi^{k+1})} \mathbb{E} \left\{ g_k(x_k, \mu_k(x_k), w_k) + \sum_{i=k+1}^{N-1} g_i(x_i, \mu_i(x_i), w_i) + g_N(x_N) \right\}$$

$$= \min_{\mu_k} \left\{ g_k(x_k, \mu_k(x_k), w_k) + \min_{\pi^{k+1}} \left[ \sum_{i=k+1}^{N-1} g_i(x_i, \mu_i(x_i), w_i) + g_N(x_N) \right] \right\}$$

by D.P principle (optimize tail subproblem then $\mu_k$)
Proof that D.P. Algorithm gives Optimal Solution

\[
= \min_{\mu_k} \mathbb{E}\{g_k(x_k, \mu_k(x_k), w_k) + J^*_{k+1}(f_k(x_k, \mu_k(x_k), w_k))\} \quad \text{by definition of } J^*_{k+1}(x_{k+1})
\]

\[
= \min_{\mu_k} \mathbb{E}\{g_k(x_k, \mu_k(x_k), w_k) + J_{k+1}(f_k(x_k, \mu_k(x_k), w_k))\} \quad \text{by induction hypothesis}
\]

\[
= \min_{u_k \in U_k(x_k)} \mathbb{E}\{g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))\} \quad \text{using fact that } \\
\min_{\mu} F(x, \mu(x)) = \min_{u \in U(x)} F(x, u).
\]

\[
= J_k(x_k) \quad \text{from D.P. algorithm equations}
\]

- So \( J^*_k(x_k) = J_k(x_k) \), and \( \mu^*_k(x_k) = u^*_k \) is the optimal policy.
- By induction, this is true for \( k = N, N - 1, \ldots, 1, 0 \).
- In particular, \( J^*(x_0) = J^*_0(x_0) = J_0(x_0) \) is the optimal cost.
Shortest Paths in a Trellis

Find shortest path from a node in Stage 1 to a node in Stage $N$

- states $\rightarrow$ nodes
- controls $\rightarrow$ arcs
- $a_{ij}^k$: cost of transition from state $i$ at stage $k$ to state $j$ at stage $k + 1$.
- $a_{it}^N$: terminal cost of state $i$
- cost function $=$ length of path from $s$ to $t$

Figure 6: Shortest paths in a trellis
Shortest Paths in a Trellis

D.P. Algorithm:

\[ J_N(i) = a_{it}^N \]
\[ J_k(i) = \min_j [a_{ij}^k + J_{k+1}(j)], \quad k = N - 1, \ldots, 1, 0 \]

Optimal cost = \( J_0(s) \) = length of shortest path from \( s \) to \( t \).

Example: Find shortest path from stage 1 to stage 3 in Figure 7.

![Figure 7: Shortest paths example](image-url)
Shortest Paths in a Trellis

Redraw as a trellis with initial and terminal node, see Figure 8.

Figure 8: Redrawn shortest paths example

Here $N = 3$.

Call the top node state 1 and bottom node state 2.

Stage $N$:

- $J_3(1) = 0$
- $J_3(2) = 0$
Shortest Paths in a Trellis

Stage 2:

- \( J_2(1) = \min\{a_{11}^2 + J_3(1), a_{12}^2 + J_3(2)\} \)
  \( = \min\{100 + 0, 200 + 0\} = 100 \)

- \( J_2(2) = \min\{a_{21}^2 + J_3(1), a_{22}^2 + J_3(2)\} \)
  \( = \min\{350 + 0, 400 + 0\} = 350 \)

Stage 1:

- \( J_1(1) = \min\{a_{11}^1 + J_2(1), a_{12}^1 + J_2(2)\} \)
  \( = \min\{300 + 100, 400 + 350\} = 400 \)

- \( J_1(2) = \min\{a_{21}^1 + J_2(1), a_{22}^1 + J_2(2)\} \)
  \( = \min\{150 + 100, 50 + 350\} = 250 \)

Stage 0:

- \( J_0(s) = \min\{0 + J_1(1), 0 + J_1(2)\} = 250 \)

Shortest path to original problem shown in red in Figure 7.
Forward D.P. Algorithm

- Observe that optimal path \( s \rightarrow t \) is also optimal path \( t \rightarrow s \) if directions of arcs are reversed.
  \[ \Rightarrow \] Shortest path algorithm can be run forwards in time (see Bertsekas for equations).
- Figure 9 shows the result of forward D.P. on shortest paths example.
- Forward D.P. useful in real-time applications, where data arrives just before you need to make a decision.
- Viterbi algorithm uses this idea
- Shortest paths is a deterministic problem, so forward D.P. works.
- For stochastic problems, no such concept of forward D.P.
  - Impossible to guarantee that any given state can be reached
Forward D.P. Algorithm

![Graph with nodes and edges labeled with numbers like 0, 150, 250, and 1]  

Figure 9: Forward D.P. on shortest paths example
Viterbi Algorithm Applications

- Estimation of hidden Markov models (HMMs)
  - $x_k =$ Markov chain
  - state transitions in $x_k$ not observed (hidden).
  - observe $z_k$, $r(z, i, j) =$ probability we observe $z$ given a transition in Markov chain $x_k$ from state $i$ to $j$.
  - Estimation problem:
    Given $Z_N = \{z_1, z_2, ..., z_N\}$, find a sequence $\hat{X}_N = \{\hat{x}_0, \hat{x}_1, ..., \hat{x}_N\}$ over all possible $\{x_0, x_1, ..., x_N\}$ that maximizes $P(X_N | Z_N)$.
    Note that $P(X_N | Z_N) = \frac{P(X_N, Z_N)}{P(Z_N)}$, and $P(Z_N)$ is “constant” given $Z_N$.
    So
    \[
    \max_{\{x_0, ..., x_N\}} P(X_N | Z_N) \longleftrightarrow \max_{\{x_0, ..., x_N\}} P(X_N, Z_N) \leftrightarrow \max_{\{x_0, ..., x_N\}} \ln P(X_N, Z_N)
    \]
Viterbi Algorithm Applications

- After some calculations (see Bertsekas), can show that problem is equivalent to:

\[
\min_{\{x_0, \ldots, x_N\}} - \ln(\pi_{x_0}) - \sum_{k=1}^{N} \ln(\pi_{x_{k-1}x_k} r(Z_k, x_{k-1}, x_k))
\]

where \(\pi_{x_0} = \) probability of initial state, \(\pi_{x_{k-1}x_k} = \) transition probabilities of Markov chain, and \(-\ln \pi_{x_0}\) and \(-\ln(\pi_{x_{k-1}x_k} r(Z_k, x_{k-1}, x_k))\) can be regarded as lengths of the different stages
\(\Rightarrow\) shortest path problem through a trellis

- Decoding of convolutional codes
- Channel equalization in presence of ISI (Inter-symbol interference)
General Shortest Path Problems

- No trellis structure
  
  e.g. Find the shortest path from each node to node 5 in Figure 10.

![Graph](image)

**Figure 10**: General shortest path problem

- Graph with $N + 1$ nodes $\{1, 2, ..., N, t\}$
- $a_{ij} =$ cost of moving from node $i$ to node $j$.
- Find the shortest path from each node $i$ to node $t$. 
General Shortest Path Problems

- Assume some $a_{ij}$’s can be negative, but cycles have non-negative length.
  - Then shortest path will not involve more than $N$ arcs.
- Reformulate as a trellis-type shortest path problem with $N$ arcs, by allowing arcs from node $i$ to itself with cost $a_{ii} = 0$
- D.P. algorithm:
  \[
  J_{N-1}(i) = a_{it} \\
  J_k(i) = \min_j \{a_{ij} + J_{k+1}(j)\}, \quad k = N - 2, \ldots, 1, 0
  \]
- This algorithm is essentially the Bellman-Ford algorithm.
- Other algorithms have also been invented, e.g. Dijkstra’s algorithm which can be used when all $a_{ij}$’s are positive.
3 Problems with Perfect State Information

- Linear Quadratic Control
- Optimal Stopping Problems
Problems with Perfect State Information

Will study some problems where analytical solutions can be obtained:

- Linear quadratic control
- Optimal stopping problems
- + others in Chapter 4 of Bertsekas
Linear Quadratic Control

(Linear) System:

\[ x_{k+1} = Ax_k + Bu_k + w_k, \quad k = 0, 1, \ldots, N - 1 \]

(Quadratic) Cost function:

\[
\mathbb{E} \left\{ \sum_{k=0}^{N-1} (x_k^T Q x_k + u_k^T R u_k) + x_N^T Q x_N \right\}
\]

Problem: Determine optimal policy to minimize cost function

- \( x_k, u_k, w_k \) are column vectors
- \( A, B, Q, R \) are matrices.
- \( w_k \) are independent and zero mean.
- \( Q \) is positive semi-definite.
- \( R \) is positive definite.
Linear Quadratic Control

Definition:
A symmetric matrix $M$ is positive semi-definite if $x^T M x \geq 0$, $\forall$ vectors $x$.
$M$ is positive definite if $x^T M x > 0$, $\forall x \neq 0$.

One characterization:
- $M$ is positive semi definite $\iff$ all eigenvalues of $M$ are $\geq 0$.
- $M$ is positive definite $\iff$ all eigenvalues of $M$ are $> 0$.

D.P. algorithm applied to this problem gives:

$$J_N(x_N) = x_N^T Q x_N$$
$$J_k(x_k) = \min_{u_k} \{ x_k^T Q x_k + u_k^T R u_k + J_{k+1}(A x_k + B u_k + w_k) \},$$

$$k = N - 1, \ldots, 1, 0.$$
Linear Quadratic Control

Turns out that minimization can be done analytically

\[ J_{N-1}(x_{N-1}) = \min_{u_{N-1}} \mathbb{E}\{ x_{N-1}^T Q x_{N-1} + u_{N-1}^T R u_{N-1} \} \]

\[ + (A x_{N-1} + B u_{N-1} + w_{N-1})^T Q (A x_{N-1} + B u_{N-1} + w_{N-1}) \}

\[ = \min_{u_{N-1}} \mathbb{E}\{ x_{N-1}^T Q x_{N-1} + u_{N-1}^T R u_{N-1} \} \]

\[ + x_{N-1}^T A^T Q A x_{N-1} + x_{N-1}^T A^T Q B u_{N-1} + x_{N-1}^T A^T Q w_{N-1} \]

\[ + u_{N-1}^T B^T Q A x_{N-1} + u_{N-1}^T B^T Q B u_{N-1} + u_{N-1}^T B^T Q w_{N-1} \]

\[ + w_{N-1}^T Q A x_{N-1} + w_{N-1}^T Q B u_{N-1} + w_{N-1}^T Q w_{N-1} \}

\[ = x_{N-1}^T (A^T Q A + Q) x_{N-1} + \mathbb{E}\{ w_{N-1}^T Q w_{N-1} \} \]

\[ + \min_{u_{N-1}} \{ u_{N-1}^T (R + B^T Q B) u_{N-1} + 2 x_{N-1}^T A^T Q B u_{N-1} \} \]
**Linear Quadratic Control**

**Digression**

Problem: \( \min_x f(x) \)

How to solve?

For unconstrained scalar problems, can differentiate and set derivative equal to 0.

e.g. \( \min_x (x - 2)^2 \), \( \frac{d}{dx} (x - 2)^2 = 2(x - 2) = 0 \Rightarrow x^* = 2. \)

- Similarly, differentiate \( u_{N-1}^T (R + B^T Q B) u_{N-1} + 2x_{N-1}^T A^T Q Bu_{N-1} \) with respect to the vector \( u_{N-1} \) and set equal to zero.

- Note that \( \frac{\partial (u^T A u)}{\partial u} = 2A u, \quad \frac{\partial (a^T u)}{\partial u} = a, \)

  where \( a \) and \( u \) are column vectors, and \( A \) is a symmetric matrix.

- Using above formulas, obtain \( 2(R + B^T Q B) u_{N-1} + 2B^T Q A x_{N-1} = 0 \)

  \( \Rightarrow u_{N-1}^* = -(R + B^T Q B)^{-1} B^T Q A x_{N-1} \)
Linear Quadratic Control

Substituting \( u^*_N = -(R + B^T QB)^{-1}B^T QA x_{N-1} \) back into expression for \( J_{N-1}(x_{N-1}) \), we obtain

\[
J_{N-1}(x_{N-1}) = x_{N-1}^T (A^T QA + Q) x_{N-1} + \mathbb{E}\{w_{N-1}^T Q w_{N-1}\} \\
+ x_{N-1}^T A^T QB (R + B^T QB)^{-1} (R + B^T QB) (R + B^T QB)^{-1} B^T QA x_{N-1} \\
- 2 x_{N-1}^T A^T QB (R + B^T QB)^{-1} B^T QA x_{N-1} \\
= x_{N-1}^T (A^T QA + Q) x_{N-1} - x_{N-1}^T A^T QB (R + B^T QB)^{-1} B^T QA x_{N-1} \\
+ \mathbb{E}\{w_{N-1}^T Q w_{N-1}\} \\
= x_{N-1}^T (A^T QA + Q - A^T QB (R + B^T QB)^{-1} B^T QA) x_{N-1} \\
+ \mathbb{E}\{w_{N-1}^T Q w_{N-1}\} \\
= x_{N-1}^T K_{N-1} x_{N-1} + \mathbb{E}\{w_{N-1}^T Q w_{N-1}\} \\
\]

with \( K_{N-1} = A^T QA + Q - A^T QB (R + B^T QB)^{-1} B^T QA \)
Linear Quadratic Control

Continuing on, can show that

\[ u_{N-2}^* = -(B^T K_{N-1} B + R)^{-1} B^T K_{N-1} A x_{N-2}, \]

and more generally (tutorial problem) that

\[ \mu_k^*(x_k) = -(B^T K_{k+1} B + R)^{-1} B^T K_{k+1} A x_k \]

where

\[ K_N = Q, \]

\[ K_k = A^T K_{k+1} A + Q - A^T K_{k+1} B (B^T K_{k+1} B + R)^{-1} B^T K_{k+1} A \]
Certainty Equivalence: Optimal policy is the same as solving problem for the deterministic system:

\[ x_{k+1} = Ax_k + Bu_k + \mathbb{E}[w_k], \]

where \( w_k \) is replaced by its expected value \( \mathbb{E}[w_k] = 0 \), i.e. the standard LQR problem.
Asymptotic Behaviour

Definition:

- A pair of matrices \((A, B)\), where \(A\) is \(n \times n\), \(B\) is \(n \times m\), is \textit{controllable} if the \(n \times nm\) matrix

\[
\begin{bmatrix}
B & AB & A^2B & \ldots & A^{n-1}B
\end{bmatrix}
\]

has full rank (all rows linearly independent).

- A pair \((A, C)\), where \(A\) is \(n \times n\), \(C\) is \(m \times n\), is \textit{observable} if \((A^T, C^T)\) is controllable.
Asymptotic Behaviour

**Theorem**
If \((A, B)\) is controllable and \(Q\) can be written as \(Q = C^T C\), where \((A, C)\) is observable, then:

1. \(K_k \to K\) as \(k \to -\infty\), with \(K\) satisfying the algebraic Riccati equation
   \[
   K = A^T KA + Q - A^T KB(B^T KB + R)^{-1}B^T KA
   \]

2. The steady state controller
   \[
   \mu^*(x_k) = Lx_k,
   \]
   where \(L = -(B^T KB + R)^{-1}B^T KA\), stabilizes the system, i.e. the eigenvalues of \(A + BL\) have magnitude \(< 1\).

**Proof:** See Bertsekas

**Note:** If \(u_k = Lx_k\), then \(x_{k+1} = Ax_k + Bu_k + w_k = (A + BL)x_k + w_k\). \(x_k\) stays “bounded” when the eigenvalues of \(A + BL\) have magnitude \(< 1\).
Other Variations

- $x_{k+1} = A_k x_k + B_k u_k + w_k$
  $A_k, B_k$ random, unknown, independent.

Optimal policy:

$$\mu^*_k(k) = -(R + \mathbb{E}\{B_k^T K_{k+1} B\})^{-1}\mathbb{E}\{B_k^T K_{k+1} A\} x_k,$$

where

$$K_N = Q,$$

$$K_k = \mathbb{E}\{A_k^T K_{k+1} A_k^T\} + Q$$

$$- \mathbb{E}\{A_k^T K_{k+1} B_k\}(\mathbb{E}\{B_k^T K_{k+1} B\} + R)^{-1}\mathbb{E}\{B_k^T K_{k+1} A\}$$

- may not have certainty equivalence
- may not have steady state solution

- $x_{k+1} = A x_k + B_k u_k + w_k$
  $B_k$ is random, independent, and is only revealed to us at time $k$.

Motivation: Wireless channels

Optimal Stopping Problems

- At each state, there is a “stop” control that stops the system, i.e., moves to and stays in a stop state.
- Pure stopping problem: if only other control is “continue”.
- For pure stopping problems, policy characterized by partition of set of states into:
  - stop region
  - continue region,
which may depend on time.
Example (Asset selling)

- A person has an asset for sale, e.g. a house.
- At each time $k = 0, 1, ..., N - 1$, person receives a random offer $w_k$ for the asset.
- Assume $w_k$'s are independent.
- Either accept $w_k$ at time $k + 1$, and invest money at interest rate $r$, or reject $w_k$ and wait for offer $w_{k+1}$.
- Must accept last offer $w_{N-1}$ at time $N$ if every previous offer was rejected.
- Find policy that maximizes (expected) revenue at the $N$-th period.
Example (Asset selling)

- **States**: If $x_k = T$: asset already sold (= stop state)
  If $x_k = w_{k-1}$: offer currently under consideration.

- **Controls**: \{accept, reject\}

- **System evolves as**:

  $$x_{k+1} = f_k(x_k, w_k, u_k)$$
  $$= \begin{cases} 
  T, & \text{if 1) } x_k = T \text{ or 2) } x_k \neq T \text{ and } u_k = \text{accept} \\
  w_k, & \text{otherwise.} 
  \end{cases}$$
Example (Asset selling)

- Rewards at time $k$:

$$g_N(x_N) = \begin{cases} 
  x_N, & \text{if } x_N \neq T; \\
  0, & \text{otherwise.}
\end{cases}$$

$$g_k(x_k, u_k, w_k) = \begin{cases} 
  (1 + r)^{N-k}x_k, & \text{if } x_k \neq T \text{ and } u_k = \text{accept}; \\
  0, & \text{otherwise.}
\end{cases}$$

- (For compound interest over $n$ years, final amount = $(1 + r)^n \times \text{initial amount}$.)
- Note: From the way the rewards are defined, $g_k$ is non-zero for only one $k \in \{0, 1, ..., N - 1\}$.
Example (Asset selling)

- Expected total reward

\[
\mathbb{E} \left[ \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N) \right]
\]

- D.P. algorithm (for reward maximization):

\[
J_N(x_N) = g_N(x_N) = \begin{cases} x_N, & \text{if } x_N \neq T; \\ 0, & \text{otherwise}. \end{cases}
\]

\[
J_k(x_k) = \max_{u_k} \mathbb{E}[g_k(x_k, u_k, w_k) + J_{k+1}(x_{k+1})]
\]
Example (Asset selling)

1. If \( x_k = T \), then \( g_k(x_k, u_k, w_k) = 0 \) and \( J_{k+1}(x_{k+1}) = 0 \), by property of \( g_k \) being non-zero for only one \( k \), and reward being incurred prior to time \( k \)

2. If \( x_k \neq T \), then

\[
\mathbb{E}[g_k(x_k, u_k, w_k) + J_{k+1}(x_{k+1})] = \begin{cases} 
(1 + r)^{N-k}x_k, & \text{if } u_k = \text{accept}; \\
0 + \mathbb{E}[J_{k+1}(w_k)], & \text{if } u_k = \text{reject}.
\end{cases}
\]

3. So

\[
J_k(x_k) = \max_{u_k} \mathbb{E}[g_k(x_k, u_k, w_k) + J_{k+1}(x_{k+1})]
\]

\[
= \begin{cases} 
\max((1 + r)^{N-k}x_k, \mathbb{E}[J_{k+1}(w_k)]), & \text{if } x_k \neq T, \\
0, & \text{if } x_k = T,
\end{cases}
\]

and optimal policy is of the form:

\[
u_k = \text{accept if } (1 + r)^{N-k}x_k > \mathbb{E}[J_{k+1}(w_k)]
\]

or \( u_k = \begin{cases} 
\text{accept, if } x_k > \frac{\mathbb{E}[J_{k+1}(w_k)]}{(1+r)^{N-k}}; \\
\text{reject, otherwise.}
\end{cases} \)
Example (Asset selling)

- Let
  \[ \alpha_k = \frac{\mathbb{E}[J_{k+1}(w_k)]}{(1 + r)^{N-k}} \]

- Can show (see Bertsekas) that \( \alpha_k \geq \alpha_{k+1} \) for all \( k \) if \( w_k \) are i.i.d.
  - Intuition: offer acceptable at time \( k \) should also be acceptable at time \( k + 1 \). See Figure 11

**Figure 11: Asset selling**
Example (Asset selling)

- Can also show that if \( w_k \) are i.i.d and \( N \to \infty \), then optimal policy “converges” to the stationary policy:

\[
    u_k = \begin{cases} 
    \text{accept,} & \text{if } x_k > \bar{\alpha} \\
    \text{reject,} & \text{if } x_k \leq \bar{\alpha}
    \end{cases}
\]

where \( \bar{\alpha} \) is constant.
General Stopping Problems

- Pure stopping problem - stop or continue only possible controls
- General stopping problem - stop or choose a control $u_k$ from $U(x_k)$ (where $U$ has more than one element)
- Consider time invariant case: $f(x_k, u_k, w_k), g(x_k, u_k, w_k)$ don't depend on $k$, and $w_k$ is i.i.d.
- Stop at time $k$ with cost $t(x_k)$
- Must stop by last stage.

D.P. algorithm:

$$J_N(x_N) = t(x_N),$$

$$J_k(x_k) = \min \{ t(x_k), \min_{u_k \in U(x_k)} \mathbb{E}\{g(x_k, u_k, w_k) + J_{k+1}(f(x_k, u_k, w_k))\} \}$$

Optimal to stop when

$$t(x_k) \leq \min_{u_k \in U(x_k)} \mathbb{E}\{g(x_k, u_k, w_k) + J_{k+1}(f(x_k, u_k, w_k))\}$$
General Stopping Problems

- Stopping set at time $k$ (set of states where you stop) defined as

$$T_k = \{x | t(x) \leq \min_{u \in U(x)} \mathbb{E}[g(x, u, w) + J_{k+1}(f(x, u, w))]\}$$

- Note that $J_{N-1}(x) \leq J_N(x)$ for all $x$, since $J_N(x) = t(x)$ and

$$J_{N-1}(x) = \min \left[ t(x), \min_{u \in U(x)} \mathbb{E}[g(x, u, w) + J_{k+1}(f(x, u, w))] \right]$$

$$\leq t(x) = J_N(x)$$

- Can show that $J_k(x) \leq J_{k+1}(x)$ (Monotonicity principle: tutorial problem)

- Then we have:

$$T_0 \subseteq T_1 \subseteq T_2 \subseteq \ldots \subseteq T_k \subseteq T_{k+1} \subseteq \ldots \subseteq T_{N-1}$$

i.e. set of states in which we stop increases with time.
Special Case

- If $f(x, u, w) \in T_{N-1}$ for all $x \in T_{N-1}, u \in U(x), w$, i.e. the set $T_{N-1}$ is absorbing, then

$$T_0 = T_1 = T_2 = \cdots = T_{N-1}.$$

Proof: See Bertsekas

- Simplifies optimal policy, called the one step lookahead policy.
Special Case

E.g. Asset selling with past offers retained

- Same situation as before, except that previously rejected offers can be accepted at a later time.
- State evolves as
  \[ x_{k+1} = \max(x_k, w_k) \]
  (instead of \( x_{k+1} = w_k \) before)
- Can show (see Bertsekas) that \( T_{N-1} = \{ x | x \geq \bar{\alpha} \} \) for some constant \( \bar{\alpha} \)
- This set is absorbing, since best currently received offer cannot decrease over time.
  \( \Rightarrow \) optimal policy at every time \( k \) is to accept if best offer \( > \bar{\alpha} \)
- Have constant threshold \( \bar{\alpha} \) even for finite horizon \( N \)
Outline

4 Problems with Imperfect State Information
- Reformulation as Perfect State Information Problem
- Linear Quadratic Control with Noisy Measurements
- Sufficient Statistics
Problems with Imperfect State Information

- State $x_k$ not known to controller.
- Instead have “noisy” observations $z_k$ of the form:

$$
\begin{align*}
    z_0 &= h_0(x_0, v_0), \\
    z_k &= h_k(x_k, u_{k-1}, v_k), \quad k = 1, 2, \ldots, N - 1,
\end{align*}
$$

where $v_k$ is “observation noise”, with a probability distribution

$$
\mathbb{P}_v(. | x_0, \ldots, x_k, u_0, \ldots, u_{k-1}, w_0, \ldots, w_{k-1}, v_0, \ldots, v_{k-1})
$$

which can depend on states, controls and disturbances

- Examples

$$
\begin{align*}
    h_x(x_k, u_{k-1}, v_k) &= x_k + v_k, \\
    h_k(x_k, u_{k-1}, v_k) &= \sin x_k + u_{k-1}v_k
\end{align*}
$$
Problems with Imperfect State Information

- Initial state $x_0$ is random with distribution $\mathbb{P}_{x_0}$.
- $u_k \in U_k$, where $U_k$ does not depend on (unknown) $x_k$.
- Information vector, i.e. information available to controller at time $k$, defined as
  \[
  I_0 = z_0, \\
  I_k = (z_0, \ldots, z_k, u_0, \ldots, u_{k-1}), k = 1, 2, \ldots, N - 1
  \]
- Policies $\pi = (\mu_0, \ldots, \mu_{N-1})$, where now $\mu_k(I_k) \in U_k$ (before $\mu_k(x_k)$).
Basic Problem with Imperfect State Information

- Find $\pi$ that minimizes the cost function

$$J_\pi = \mathbb{E}\left\{ \sum_{k=0}^{N-1} g_k(x_k, \mu_k(l_k), w_k) + g_N(x_N) \right\}$$

s.t. system equation

$$x_{k+1} = f_k(x_k, \mu_k(l_k), w_k)$$

and measurement equation

$$z_k = h_k(x_k, \mu_{k-1}(l_{k-1}), v_k)$$

- **Question:** How to solve this problem?
Reformulation as Perfect State Information Problem

- **Idea:** Define new system where the state is $l_k$. Then have D.P. algorithm etc.
- By definition

$$l_{k+1} = (z_0, \ldots, z_k, z_{k+1}, u_0, \ldots, u_{k-1}, u_k)$$

$$= (z_0, \ldots, z_k, u_0, \ldots, u_{k-1}, z_{k+1}, u_k)$$

$$l_k$$

$$\Rightarrow l_{k+1} = (l_k, u_k, z_{k+1}).$$
Reformulation as Perfect State Information Problem

- Regard
  \[ I_{k+1} = (I_k, u_k, z_{k+1}) \]
  as a dynamical system with state \( I_k \), control \( u_k \) and disturbance \( z_{k+1} \).

- Next note that \( \mathbb{E}[g_k(x_k, u_k, w_k)] = \mathbb{E}[\mathbb{E}[g_k(x_k, u_k, w_k)|l_k, u_k]] \) (Recall that \( \mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|Y]] \)).

- Define \( \tilde{g}_k(I_k, u_k) = \mathbb{E}[g_k(x_k, u_k, w_k)|I_k, u_k] = \) cost per stage of new system, and \( \tilde{g}_N(I_N) = \mathbb{E}[g_N|I_N] = \) terminal cost.

- Cost function becomes

\[
\mathbb{E} \left\{ \sum_{k=0}^{N-1} g_k(x_k, \mu_k(I_k), w_k) + g_N(x_N) \right\} = \mathbb{E} \left\{ \sum_{k=0}^{N-1} \tilde{g}_k(I_k, \mu_k(I_k)) + \tilde{g}_N(I_N) \right\}
\]
Reformulation as Perfect State Information Problem

- D.P. algorithm for reformulated perfect state information problem is:

\[ J_N(I_N) = \tilde{g}_N(I_N) = \mathbb{E}[g_N(x_N)|I_N] \]

\[ J_k(I_k) = \min_{u_k \in U_k} \mathbb{E}\{\tilde{g}_k(I_k, u_k) + J_{k+1}(I_k, u_k, z_{k+1})\} \]

\[ = \min_{u_k \in U_k} \mathbb{E}\{g_k(x_k, u_k, w_k) + J_{k+1}(I_k, u_k, z_{k+1})|I_k\}, \quad k = N - 1, \ldots \]

- Optimal cost \( J^* = \mathbb{E}\{J_0(z_0)\} \)
Linear Quadratic Control with Noisy Measurements

- **System**
  \[ x_{k+1} = Ax_k + Bu_k + w_k \]

- **Cost function**
  \[
  \mathbb{E} \left[ \sum_{k=0}^{N-1} \left( x_k^T Q x_k + u_k^T R u_k \right) + x_N^T Q x_N \right] + g_k(x_k, u_k, w_k)
  \]

- **Observations**
  \[ z_k = Cx_k + v_k \]

- \( w_k \) are independent, zero mean.
- **From D.P. Algorithm:**
  \[ J_N(I_N) = \mathbb{E}[x_N^T Q x_N | I_N], \]
Linear Quadratic Control with Noisy Measurements

\[ J_{N-1}(I_{N-1}) = \min_{u_{N-1}} \mathbb{E}\left\{ x_{N-1}^T Q x_{N-1} + u_{N-1}^T R u_{N-1} \right. \]

\[ + \mathbb{E}\left[ (Ax_{N-1} + Bu_{N-1} + w_{N-1})^T Q (Ax_{N-1} + Bu_{N-1} + w_{N-1}) \middle| I_{N} \right] \left| I_{N-1} \right\} \]

\[ = \min_{u_{N-1}} \mathbb{E}\left\{ x_{N-1}^T Q x_{N-1} + u_{N-1}^T R u_{N-1} \right. \]

\[ + \left. (Ax_{N-1} + Bu_{N-1} + w_{N-1})^T Q (Ax_{N-1} + Bu_{N-1} + w_{N-1}) \middle| I_{N-1} \right\} \]

(Using the tower property that \( \mathbb{E}(\mathbb{E}(X|Y)|Z) = \mathbb{E}(X|Z) \) if \( Y \) contains “more information” than \( Z \))

\[ = \ldots \text{ (expand, simplify and use } \mathbb{E}(w_{N-1}|I_{N-1}) = 0.) \]

\[ = \mathbb{E}\left[ x_{N-1}^T (A^T Q A + Q) x_{N-1} \middle| I_{N-1} \right] + \mathbb{E}\left[ w_{N-1}^T Q w_{N-1} \middle| I_{N-1} \right] \]

\[ + \min_{u_{N-1}} \left\{ u_{N-1}^T (B^T Q B + R) u_{N-1} + 2 \mathbb{E}\left[ x_{N-1} \middle| I_{N-1} \right]^T A^T Q B u_{N-1} \right\} \]

Differentiate with respect to \( u_{N-1} \) and set equal to zero:

\[ 2(B^T Q B + R) u_{N-1} + 2B^T Q Ax_{N-1} = 0 \]

\[ \Rightarrow u_{N-1}^* = -(B^T Q B + R)^{-1} B^T Q A \mathbb{E}[x_{N-1}|I_{N-1}] \]
Linear Quadratic Control with Noisy Measurements

Substituting expression for $u^*_N$ back in:

$$J_{N-1}(I_{N-1}) = \mathbb{E}[x_{N-1}^T(A^T QA + Q)x_{N-1}|I_{N-1}] + \mathbb{E}[w_{N-1}^T Qw_{N-1}]$$
$$+ \mathbb{E}[x_{N-1}|I_{N-1}]^T A^T QB(B^T QB + R)^{-1}(B^T QB + R)$$
$$\times (B^T QB + R)^{-1}B^T QA\mathbb{E}[x_{N-1}|I_{N-1}] - 2\mathbb{E}[x_{N-1}|I_{N-1}]^T$$
$$\times A^T QB(B^T QB + R)^{-1}B^T QA\mathbb{E}[x_{N-1}|I_{N-1}]$$
$$= \mathbb{E}[x_{N-1}^T(A^T QA + Q)x_{N-1}|I_{N-1}] + \mathbb{E}(w_{N-1}^T Qw_{N-1})$$
$$- \mathbb{E}(x_{N-1}|I_{N-1})^T A^T QB(B^T QB + R)^{-1}B^T QA\mathbb{E}(x_{N-1}|I_{N-1})$$
$$= \mathbb{E}[x_{N-1}^T(A^T QA + Q)x_{N-1}|I_{N-1}] + \mathbb{E}(w_{N-1}^T Qw_{N-1})$$
$$+ \mathbb{E}\left[(x_{N-1} - \mathbb{E}[x_{N-1}|I_{N-1}])^T A^T QB(B^T QB + R)^{-1}ight.$$ 
$$\times B^T QA(x_{N-1} - \mathbb{E}[x_{N-1}|I_{N-1}])|I_{N-1}\right]$$
$$- \mathbb{E}[x_{N-1}^T A^T QB(B^T QB + R)^{-1}B^T QA x_{N-1}|I_{N-1}]$$
$$\underbrace{P_{N-1}}$$
Linear Quadratic Control with Noisy Measurements

We have

$$J_{N-1}(I_{N-1}) = \mathbb{E}[x_{N-1}^T K_{N-1} x_{N-1} | I_{N-1}] + \mathbb{E}[w_{N-1}^T Q w_{N-1}]$$
$$+ \mathbb{E}[(x_{N-1} - \mathbb{E}[x_{N-1} | I_{N-1}])^T P_{N-1} (x_{N-1} - \mathbb{E}(x_{N-1} | I_{N-1})) | I_{N-1}]$$

where

$$P_{N-1} = A^T Q B (B^T Q B + R)^{-1} B^T Q A$$
$$K_{N-1} = A^T Q A + Q - P_{N-1}.$$
Linear Quadratic Control with Noisy Measurements

For period $N - 2$,

$$J_{N-2}(I_{N-2}) = \min_{u_{N-2}} E\{x_{N-2}^T Q x_{N-2} + u_{N-2}^T R u_{N-2} + J_{N-1}(I_{N-1})|I_{N-2}\}$$

$$= E\{x_{N-2}^T Q x_{N-2} | I_{N-2}\} + \min_{u_{N-2}} \left[ u_{N-2}^T R u_{N-2} + E\{x_{N-1}^T K_{N-1} x_{N-1} | I_{N-2}\} \right]$$

$$+ E\left[ (x_{N-1} - E[x_{N-1}|I_{N-1}])^T P_{N-1} (x_{N-1} - E[x_{N-1}|I_{N-1}]) | I_{N-2} \right]$$

$$+ E(\omega_{N-1}^T Q \omega_{N-1})$$

Then can obtain

$$u_{N-2}^* = -(B^T K_{N-1} B + R)^{-1} B^T K_{N-1} A E[x_{N-2} | I_{N-2}]$$

Note that in the above the term

$$E\left[ (x_{N-1} - E[x_{N-1}|I_{N-1}])^T P_{N-1} (x_{N-1} - E[x_{N-1}|I_{N-1}]) | I_{N-2} \right]$$

can be taken outside the minimization (see Bertsekas for proof).

Intuition: estimation error $x_k - E[x_k | I_k]$ can’t be influenced by choice of control.
Linear Quadratic Control with Noisy Measurements

Continuing on, general solution is:

\[ \mu^*_k(l_k) = u^*_k = -(B^T K_{k+1} B + R)^{-1} B^T K_{k+1} A \mathbb{E}[x_k | l_k] = L_k \mathbb{E}[x_k | l_k] \]

where

\[ K_N = Q \]
\[ P_k = A^T K_{k+1} B (B^T K_{k+1} B + R)^{-1} B^T K_{k+1} A \]
\[ K_k = A^T K_{k+1} A + Q - P_k \]

Comparison with perfect state information case:

- \( L_k \) matrix the same
- \( x_k \) is replaced by \( \mathbb{E}[x_k | l_k] \)

How to compute \( \mathbb{E}[x_k | l_k] \)?
Summary so far:

- **System**
  \[ x_{k+1} = Ax_k + Bu_k + w_k \]
  \[ z_k = Cx_k + v_k \]

- **Problem**
  \[
  \min \mathbb{E} \left[ \sum_{k=0}^{N-1} (x_k^T Q x_k + u_k^T R u_k) + x_N^T Q x_N \right]
  \]

- **Optimal solution is**
  \[
  \mu_k^*(l_k) = -(B^T K_{k+1} B + R)^{-1} B^T K_{k+1} \mathbb{E}[x_k | l_k] = L_k \mathbb{E}[x_k | l_k]
  \]
  where \( l_k = (z_0, \ldots, z_k, u_0, \ldots, u_{k-1}) \)
Linear Quadratic Control with Noisy Measurements

- Optimal controller can be decomposed into two parts:
  
  1) An estimator which computes $E[x_k | I_k]$.
  
  2) An actuator which multiplies $E[x_k | I_k]$ with $L_k$. $L_k$ is the same gain matrix as in the perfect state information case, only replace $x_k$ with $E[x_k | I_k]$.

- Estimator and actuator can be designed separately.
  - Known as the separation principle/theorem
LQG Control

- Remaining problem: How do we compute $\mathbb{E}[x_k|I_k]$?
- Very difficult problem in general (subject called non-linear filtering).
- When system is linear and $w_k, v_k$ are Gaussian, $\mathbb{E}[x_k|I_k]$ can be computed analytically.
  - Procedure/algorithmm is known as the **Kalman Filter** (ref: Anderson and Moore, “Optimal Filtering”), and the overall controller is called the **LQG** (linear quadratic Gaussian) controller.
Kalman Filter

- System:

\[ x_{k+1} = Ax_k + Bu_k + w_k \]
\[ z_k = Cx_k + v_k \]

\[ w_k \sim N(0, \Sigma_w) \text{ i.i.d.} \quad \Sigma_w = \mathbb{E}[w_k w_k^T] \]
\[ v_k \sim N(0, \Sigma_v) \text{ i.i.d.} \quad \Sigma_v = \mathbb{E}[v_k v_k^T] \]

- Define state estimates

\[ \hat{x}_{k|k} = \mathbb{E}[x_k|I_k] \]
\[ \hat{x}_{k+1|k} = \mathbb{E}[x_{k+1}|I_k] \]

and estimation error covariance matrices

\[ \Sigma_{k|k} = \mathbb{E}[(x_k - \hat{x}_{k|k})(x_k - \hat{x}_{k|k})^T|I_k] \]
\[ \Sigma_{k+1|k} = \mathbb{E}[(x_{k+1} - \hat{x}_{k+1|k})(x_{k+1} - \hat{x}_{k+1|k})^T|I_k] \]
Kalman Filter

Then $\hat{x}_{k|k}, \hat{x}_{k+1|k}, \Sigma_{k|k}, \Sigma_{k+1|k}$ can be computed recursively using the Kalman Filter equations:

\[
\hat{x}_{k|k} = \hat{x}_{k|k-1} + \Sigma_{k|k-1} C^T (C \Sigma_{k|k-1} C^T + \Sigma_v)^{-1} (z_k - C \hat{x}_{k|k-1}) \\
\hat{x}_{k+1|k} = A \hat{x}_{k|k} + B u_k \\
\Sigma_{k|k} = \Sigma_{k|k-1} - \Sigma_{k|k-1} C^T (C \Sigma_{k|k-1} C^T + \Sigma_v)^{-1} C \Sigma_{k|k-1} \\
\Sigma_{k+1|k} = A \Sigma_{k|k} A^T + \Sigma_w, \quad k = 0, 1, \ldots, N - 1
\]

Proof: see Bertsekas, or Anderson and Moore.

Beware: Many people who work in Kalman filtering like to use $Q$ for $\Sigma_w$, $R$ for $\Sigma_v$, $K_k$ for the “Kalman gain” $\Sigma_{k|k-1} C^T (C \Sigma_{k|k-1} C^T + \Sigma_v)^{-1}$, but here $Q, R, K_k$ have been used for different things. People also use $P_{k+1|k}$ for $\Sigma_{k+1|k}$, $P_{k|k}$ for $\Sigma_{k|k}$ etc.
Kalman Filter Properties

- In general the mean squared error

\[ \mathbb{E}[(x_k - \hat{x}_k)^T(x_k - \hat{x}_k)|I_k] \]

is minimized when \( \hat{x}_k = \mathbb{E}[x_k|I_k] \)

- Kalman filter equations compute \( \mathbb{E}[x_k|I_k] \) when noises are Gaussian, and (optimal) estimates are linear functions of the measurements \( z_k \).

- Even when noises are not Gaussian, \( \hat{x}_{k|k} \) computed by Kalman filter equations gives the best linear estimate of \( x_k \).
  - Useful suboptimal solution when noises are non-Gaussian.
Kalman Filter Properties

- Recall that if the pair \((A, B)\) is controllable and \((A, Q^{1/2})\) is observable, optimal controller has a steady state solution.

- Similarly, if \((A, C)\) is observable, and \((A, \Sigma^{1/2})\) is controllable, then \(\Sigma_{k|k-1}\) converges to a steady state value \(\Sigma\) as \(k \to \infty\), where \(\Sigma\) satisfies the algebraic Riccati equation

\[
\Sigma = A\Sigma A^T - A\Sigma C^T (C\Sigma C^T + \Sigma_v)^{-1} C\Sigma A^T + \Sigma_w
\]

- So we have a steady state estimator:

\[
\hat{x}_{k|k} = \hat{x}_{k|k-1} + \Sigma C^T (C\Sigma C^T + \Sigma_v)^{-1}(z_k - C\hat{x}_{k|k-1}) \\
\hat{x}_{k+1|k} = A\hat{x}_{k|k} + Bu_k
\]
Sufficient Statistics

Information vector $I_k = (z_0, \ldots, z_k, u_0, \ldots, u_{k-1})$

- Dimension of $I_k$ increases with time $k$.
- Inconvenient for large $k$

**Sufficient statistic:** function $S_k(I_k)$ which summarizes all essential content in $I_k$ for computing the optimal control, i.e. $\mu^*_k(I_k) = \bar{\mu}(S_k(I_k))$ for some function $\bar{\mu}$.

- $S_k(I_k)$ preferably of smaller dimension than $I_k$. 
Examples of Sufficient Statistics

1) $I_k$ itself

2) Conditional state distribution/belief state $\mathbb{P}_{x_k|I_k}$, assuming that distribution of $v_k$ depends only on $x_{k-1}, u_{k-1}, w_{k-1}$.
   - If number of states is finite then $\mathbb{P}_{x_k|I_k}$ is a vector.
   - e.g. if states are 1, 2, ..., $n$, then
     \[
     \mathbb{P}_{x_k|I_k} = \begin{bmatrix}
     \mathbb{P}(x_k = 1|I_k) \\
     \mathbb{P}(x_k = 2|I_k) \\
     \vdots \\
     \vdots \\
     \mathbb{P}(x_k = n|I_k)
     \end{bmatrix}
     \]
     Dimension of vector is $n$, which doesn’t grow with $k$

3) Special case: $\mathbb{E}[x_k|I_k]$ is a sufficient statistic for LQG problem (though not a sufficient statistic in general).
For conditional state distribution, $P_{x_k|I_k}$ can be generated recursively, as

$$P_{x_{k+1}|I_{k+1}} = \Phi_k(P_{x_k|I_k}, u_k, z_{k+1})$$

for some function $\Phi_k(\cdot, \cdot, \cdot)$.

Then D.P. algorithm can be written as

$$\bar{J}_k(P_{x_k|I_k}) = \min_{u_k \in U_k} \mathbb{E}[g_k(x_k, u_k, w_k) + \bar{J}_{k+1}(\Phi_k(P_{x_k|I_k}, u_k, z_{k+1}))|I_k].$$

General formula for $\Phi_k(\cdot, \cdot, \cdot)$ can be derived, but is quite complicated (see Bertsekas). Will derive some examples from first principles.
Example 1: Search Problem

- At each period, decide whether to search a site that may contain a treasure.
- If treasure is present and we search, we find it with probability $\beta$ and take it.
- States: \{treasure present, treasure not present\}
- Controls: \{search, no search\}
- Regard each search result as (imperfect) observation of the state.
- Let $p_k =$ probability treasure present at start of time $k$.
  - If not search, $p_{k+1} = p_k$.
  - If search and find treasure, $p_{k+1} = 0$. 
Example 1

- If search and don’t find treasure,

\[
p_{k+1} = \mathbb{P}(\text{treasure present at } k | \text{don’t find at } k)
\]

\[
= \frac{\mathbb{P}(\text{treasure present at } k \cap \text{don’t find at } k)}{\mathbb{P}(\text{don’t find at } k)}
\]

\[
= \frac{p_k (1 - \beta)}{p_k (1 - \beta) + (1 - p_k)},
\]

with \((1 - p_k)\) corresponding to treasure not present & don’t find.

- Thus

\[
p_{k+1} = \begin{cases} 
  p_k, & \text{not search at time } k \\
  0, & \text{search and find treasure.} \\
  \frac{p_k (1 - \beta)}{p_k (1 - \beta) + (1 - p_k)}, & \text{search and don’t find treasure}
\end{cases}
\]

\[
= \Phi_k(p_k, u_k, z_{k+1}) \text{ function.}
\]
Example 1

- Now let treasure be worth $V$, each search costs $C$, and once we decide not to search we can’t search again at future times.
- D.P. algorithm gives:

$$
\bar{J}_k(p_k) = \max \begin{cases} 
\text{no search, search} \\
0, -C + p_k \beta V \\
(1 - p_k \beta) \bar{J}_{k+1} \left( \frac{p_k (1 - \beta)}{p_k (1 - \beta) + 1 - p_k} \right) + p_k \beta \bar{J}_{k+1}(0)
\end{cases}
$$

$$
= \max \begin{cases} 
\text{no search, search} \\
0, -C + p_k \beta V \\
(1 - p_k \beta) \bar{J}_{k+1} \left( \frac{p_k (1 - \beta)}{p_k (1 - \beta) + 1 - p_k} \right)
\end{cases}
$$

(where $p_k \beta \bar{J}_{k+1}(0) = 0$ since treasure already found)

- Can show that $\bar{J}_k(p_k) = 0, \forall p_k \leq \frac{C}{\beta V}$, and that it is optimal to search iff expected reward $p_k \beta V \geq$ cost of search $C$. (Tutorial problem)
Example 2: Research Paper*

A process \( \{P_{e,k}\} \) evolves in the following way, for \( k = 1, \ldots, N \):

\[
P_{e,k+1} = \begin{cases} 
\bar{P}, & \nu_{k+1}\gamma_{e,k+1} = 1 \\
AP_{e,k}A^T + Q, & \nu_{k+1}\gamma_{e,k+1} = 0,
\end{cases}
\]

- \( \bar{P}, A, Q \) are some matrices
- \( \gamma_{e,k} \) is i.i.d Bernoulli process with
  \[
  \mathbb{P}(\gamma_{e,k} = 1) = \lambda_{e}, \mathbb{P}(\gamma_{e,k} = 0) = 1 - \lambda_{e}, \forall k
  \]
- \( \nu_k \in \{0, 1\} \)
- \( \{P_{e,k}\} \) is not observed at all (no observation \( z_k \)).

Example 2

- Regard $P_{e,k}$ as the state at time $k$, and $\nu_{k+1}$ as the control. Assume $P_{e,0} = \bar{P}$

- Then $P_{e,k} \in \{\bar{P}, A\bar{P}A^T + Q, A(A\bar{P}A^T + Q) + Q,...\} = \{\bar{P}, f(\bar{P}), f^2(\bar{P}), ..., f^N(\bar{P})\}$, where

$$f(\bar{P}) = A\bar{P}A^T + Q$$

- Conditional state distribution is

$$\begin{bmatrix}
P(P_{e,k} = \bar{P}|\nu_0, ..., \nu_k) \\
P(P_{e,k} = f(\bar{P})|\nu_0, ..., \nu_k) \\
\vdots \\
P(P_{e,k} = f^N(\bar{P})|\nu_0, ..., \nu_k)
\end{bmatrix}$$
Example 2

When $\nu_{k+1} = 0$, $P_{e,k+1} = f(P_{e,k})$ with probability 1. So

\[
\begin{bmatrix}
\mathbb{P}(P_{e,k+1} = \bar{P}|\nu_0, \ldots, \nu_{k+1}) \\
\mathbb{P}(P_{e,k+1} = f(\bar{P})|\nu_0, \ldots, \nu_{k+1}) \\
\vdots \\
\mathbb{P}(P_{e,k+1} = f^{N-1}(\bar{P})|\nu_0, \ldots, \nu_{k+1})
\end{bmatrix}
= \begin{bmatrix}
0 \\
\mathbb{P}(P_{e,k} = \bar{P}|\nu_0, \ldots, \nu_k) \\
\vdots \\
\mathbb{P}(P_{e,k} = f^{N-1}(\bar{P})|\nu_0, \ldots, \nu_k)
\end{bmatrix}
\]

$\Phi_k(\mathbb{P}_{P_{e,k}|l_k, \nu_{k+1}, z_{k+1}})$ function when $\nu_{k+1} = 0$
Example 2

- When $\nu_{k+1} = 1$, $P_{e,k+1} = \bar{P}$ with probability $\lambda_e$, and $P_{e,k+1} = f(P_{e,k})$ with probability $1 - \lambda_e$. So

$$\begin{bmatrix}
\mathbb{P}(P_{e,k+1} = \bar{P}|\nu_0, \ldots, \nu_{k+1}) \\
\mathbb{P}(P_{e,k+1} = f(\bar{P})|\nu_0, \ldots, \nu_{k+1}) \\
\vdots \\
\mathbb{P}(P_{e,k+1} = f^N(\bar{P})|\nu_0, \ldots, \nu_{k+1})
\end{bmatrix}
= \begin{bmatrix}
\lambda_e \\
(1 - \lambda_e)\mathbb{P}(P_{e,k} = \bar{P}|\nu_0, \ldots, \nu_k) \\
\vdots \\
(1 - \lambda_e)\mathbb{P}(P_{e,k} = f^{N-1}(\bar{P})|\nu_0, \ldots, \nu_k)
\end{bmatrix}
= \Phi_k(\mathbb{P}P_{e,k}|I_k, \nu_{k+1}, z_{k+1}) \text{ function when } \nu_{k+1} = 1
Outline

5. Suboptimal Methods / Approximate Dynamic Programming
   - Certainty Equivalent Control
   - Rollout Algorithms
   - Model Predictive Control
Suboptimal Methods

Why do we need/want suboptimal methods?

- In D.P. need to compute

\[ J_k(x_k) = \min_u \mathbb{E}[g_k(x_k, u, w_k) + J_{k+1}(x_{k+1})] \]

for all states \( x_k \)

1) In many problems, this minimization can’t be done analytically.

- Have to test each \( u_k \).
- When number of possible \( x_k, u_k \) or \( w_k \) are large, amount of computation required can be substantial.
Suboptimal Methods

2) In some problems, $x_k$, $u_k$ or $w_k$ are continuous valued.

- Have to discretize their ranges to convert to discrete problem, see Fig. 12.
- Using more points gives better approximation, but more computation required.
- Situation worse for higher dimensions - “curse of dimensionality”.

![Discretization diagram](image)

Range of $x_k = [-1,1]$

Figure 12: Discretization
Suboptimal Methods

3) In problems with imperfect state information, conditional state distribution $P_{x_k|I_k}$ is of the form

$$
P_{x_k|I_k} = \begin{bmatrix}
P(x_k = 1|I_k) \\
P(x_k = 2|I_k) \\
... \\
... \\
... \\
P(x_k = n|I_k)
\end{bmatrix}$$

- Range is $[0, 1]^n$ (continuous).
- Solving imperfect state information problems exactly is intractable except in special or very simple cases.

4) Real time constraints, data not available until shortly before, or data may change as system is being controlled.
Suboptimal Methods

Will discuss a few methods for suboptimal solutions

- Certainty Equivalent Control (CEC)
- Rollout Algorithms
- Model Predictive Control (MPC)
- Many other methods in Vol. I Ch.6 and Vol. II of Bertsekas.
Certainty Equivalent Control

Idea

- Replace a stochastic problem with a deterministic one
- At each time $k$, fix the future uncertain quantities to some “typical” values, e.g. replace $w_k$ with $\mathbb{E}[w_k]$.

Procedure (Online Version)

At each time $k$

1. Fix $w_i, i \geq k$ to some $\bar{w}_i$. Solve the deterministic problem

$$\min_{\{u_k, u_{k+1}, \ldots, u_{N-1}\}} \left[ \sum_{i=k}^{N-1} g_i(x_i, u_i, \bar{w}_i) + g_N(x_N) \right],$$

assuming $x_{i+1} = f_i(x_i, u_i, \bar{w}_i), i = k, k+1, \ldots, N-1, u_i \in U_i(x_i)$

2. Use the first control in optimal control sequence $\{\bar{u}_k, \bar{u}_{k+1}, \ldots, \bar{u}_{N-1}\}$ found, i.e. $\bar{\mu}_k(x_k) = \bar{u}_k$
Certainty Equivalent Control

Equivalent Procedure (Offline Version)

(1) Fix \( w_k \) to some \( \bar{w}_k \) for \( k = 0,1,..,N - 1 \). Solve the deterministic problem

\[
\min_{\{\mu_0,\mu_1,\ldots,\mu_{N-1}\}} \left[ \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), \bar{w}_k) + g_N(x_N) \right],
\]

assuming \( x_{k+1} = f_k(x_k, \mu_k(x_k), \bar{w}_k), k = 0,1,\ldots,N-1, u_k(x_k) \in U_k(x_k) \)

(2) Let \( \{\mu_0^d, \mu_1^d, \ldots, \mu_{N-1}^d\} \) be the solution to problem above. At each time \( k \), apply \( \bar{\mu}_k(x_k) = \mu_k^d(x_k) \)
Certainty Equivalent Control

Comments:

- $N$ problems have to be solved in online version, one in the offline version.

- Online and offline versions give same controller if data is not changing. Use online version if data is changing.

- For problems with imperfect state information, also replace $x_k$ by estimate $\bar{x}_k(I_k)$ (e.g. $\bar{x}_k(I_k) = \mathbb{E}[x_k|I_k]$).

- Certainty Equivalent Control often performs well in practice.

- For linear quadratic control problem, Certainty Equivalent Controller is equivalent to optimal controller.

- Can fix some disturbances while leaving others stochastic, e.g. for imperfect state information problems, replace $x_k$ by $\bar{x}_k(I_k)$ while leaving $w_k$ as stochastic.
Rollout Algorithms

One step lookahead policy, with optimal cost to go approximated by cost to go of some base policy.

- “Rollout” coined by Gerald Tesauro in 1996 in the context of rolling dice in a backgammon playing computer program.
- A given backgammon position evaluated by “rolling out” many games starting from that position, and taking average.
- Rollout policy has a cost improvement property
- Often produces substantial improvement over base policy.
Rollout Algorithms

One step lookahead policy

- At each $k$ and $x_k$ we use the control $\mu_k(x_k)$ that solves the problem

\[
\min_{u_k \in U_k} \mathbb{E}\{g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k))\}
\]

where $\tilde{J}_N = g_N$, and $\tilde{J}_{k+1}$ is approximation to true cost to go $J_{k+1}$.

Rollout policy

- When the approximation $\tilde{J}_k$ is cost to go of some heuristic base policy.
Example: Quiz problem

- \( N \) questions given.
- Question \( i \) answered correctly with probability \( p_i \), reward \( v_i \) if correct.
- Quiz terminates at first incorrect answer.
- Choose order of questions to maximize total reward.

- Index policy: answer questions in decreasing order of \( \frac{p_i v_i}{1 - p_i} \)
  - Index policy is optimal when no other constraints (Ch. 4.5 Bertsekas).
- Now assume there is a limit (< \( N \)) on maximum number of questions to be answered.
  - Then index policy in general is not optimal.
Example: Quiz problem

- Rollout algorithm: use index policy as base policy.
  - At a state denoting the subset of questions already answered, compute the expected reward $R(j)$ for each possible next question $j$, assuming the order of remaining questions follows index policy.
  - Answer the question with maximum $R(j)$.

- $R(j)$ can be computed analytically, since given an order of questions $(i_1, i_2, ..., i_M)$, with $M \leq N$, the expected reward is

$$p_{i_1}(v_{i_1} + p_{i_2}(v_{i_2} + p_{i_3}(... + p_{i_M}v_{i_M})...))$$
Example: Travelling Salesman Problem

- \( N \) cities
- Assume graph is complete
- Find the minimum cost tour that visits each city exactly once and returns to the city you started from.
- Important and difficult problem in combinatorial optimization.

![Travelling Salesman Problem Diagram]

Figure 13: Travelling Salesman Problem
Example: Travelling Salesman Problem

- Nearest neighbour heuristic:
  - Start from an arbitrary city.
  - Next city visited is the one with minimum distance from current city (and has not been previously visited)

- Rollout algorithm: Use the nearest neighbour heuristic as base policy.
  - For each node not yet visited, assume nearest neighbour heuristic is then run afterwards, and compute cost of the tour.
  - Choose next city as the one that gives best tour.
Example: Travelling Salesman Problem

- Consider the travelling salesman problem for the graph shown below.
- Let $a$ be the node with which we start and end the tour. An optimal tour can be shown to be $abcdea$, with length 375.

![Graph of the Travelling Salesman Problem](image.png)

**Figure 14: Travelling Salesman Problem**

- Nearest neighbour ($N.N.$) heuristic gives tour $aedcba$ with length 550.
Example: Travelling Salesman Problem

- Rollout algorithm with nearest neighbour heuristic as base policy:

1st stage:

- **ab cdea**  \( \text{length} = 375 \)
- **ac bdea**  \( \text{length} = 550 \)
- **ad ebca**  \( \text{length} = 625 \)
- **ae dbca**  \( \text{length} = 550 \)

So next node should be **b**.

2nd stage:

- **abc dea**  \( \text{length} = 375 \)
- **abd eca**  \( \text{length} = 650 \)
- **abe dca**  \( \text{length} = 675 \)

So next node is **c**
Example: Travelling Salesman Problem

- Rollout algorithm with nearest neighbour heuristic as base policy:
  3rd stage:
  \[ abcd \quad ea \quad \text{length} = 375 \]
  \[ \underbrace{abce \quad da}_{N.N.} \quad \text{length} = 425 \]
  So next node is \( d \)
  4th stage:
  \[ abcdea = \text{tour computed by rollout, with length 375} \]
Cost Improvement Property of Rollout Algorithm

**Theorem:**
Let $\bar{J}_k(x_k)$ be the cost to go of rollout policy. Let $\tilde{J}_k(x_k)$ be the cost to go of base policy. Then

$$\bar{J}_k(x_k) \leq \tilde{J}_k(x_k), \forall x_k, k$$

**Proof:** Use induction

Initial step:
- By definition

$$\bar{J}_N(x_N) = \tilde{J}_N(x_N) = g_N(x_N), \forall x_N$$
Cost Improvement Property of Rollout Algorithm

Induction step:

- Assume \( \bar{J}_l(x_l) \leq \tilde{J}_l(x_l), \forall x_l, l = N - 1, N - 2, \ldots, k + 1 \)
- Want to show \( \bar{J}_k(x_k) \leq \tilde{J}_k(x_k) \).
- Let \( \bar{\mu}_k(x_k) \) be control applied by rollout policy.
  Let \( \tilde{\mu}_k(x_k) \) be control applied by base policy.
  
- Then
  \[
  \bar{J}_k(x_k) = \mathbb{E}[g_k(x_k, \bar{\mu}_k(x_k), w_k) + \bar{J}_{k+1}(f_k(x_k, \bar{\mu}_k(x_k), w_k))] \quad \text{by definition of cost to go function } \bar{J}_k
  \]
  \[
  \leq \mathbb{E}[g_k(x_k, \tilde{\mu}_k(x_k), w_k) + \bar{J}_{k+1}(f_k(x_k, \tilde{\mu}_k(x_k), w_k))] \quad \text{by induction hypothesis}
  \]
  \[
  \leq \mathbb{E}[g_k(x_k, \tilde{\mu}_k(x_k), w_k) + \tilde{J}_{k+1}(f_k(x_k, \tilde{\mu}_k(x_k), w_k))] \quad \text{by definition of } \tilde{\mu}_k(x_k) \text{ being optimal for rollout}
  \]
  \[
  = \tilde{J}_k(x_k) \quad \text{by definition of cost to go function } \tilde{J}_k
  \]

By induction, \( \bar{J}_k(x_k) \leq \tilde{J}_k(x_k), \forall k, x_k \).
Difficulties In Using Rollout

- For stochastic problems, cost to go $\tilde{J}_k$ of base policy may still be difficult to evaluate analytically.

- Need to approximate $\tilde{J}_k$ using e.g. Monte Carlo simulations, or certainty equivalence.
Model Predictive Control (MPC)

- Originated and widely used in process control industries.
- Concepts have since been applied to many areas.

Idea:
- Compute a set of $m$ control signals which optimizes objective over a finite horizon $m$, using a model that predicts system outputs at future times.
- First element of this set is applied to system.
- Repeat process at next time step in a receding horizon manner.
Figure 15: Model Predictive Control:
A: Computed set of control signals at time 0.
B: Computed set of control signals at time 1.
C: Computed set of control signals at time 2.
Model Predictive Control (MPC)

- Well suited to systems with control/state constraints, non-linear systems etc...
- Corresponds to a $m$-step lookahead policy with cost to go approximation equal to zero.
- As $m$ increases, performance “usually” improves (see Bertsekas for counter-examples)
- Higher amount of computations required for larger $m$.
- For nonlinear systems, computation of control signals often needs to be done numerically.
6 Infinite Horizon Problems
   - Discounted Cost Problems
   - Average Cost Problems
Infinite Horizon Problems

- Infinite number of stages.
- Assume system is stationary, i.e. $f(.,.,.)$, $g(.,.,.)$ and distribution of $w_k$ don’t depend on time $k$.
- “Different” algorithms needed.
- Optimal policies often have a simple stationary form that does not depend on time.
- But analysis is more difficult than finite horizon problems (won’t cover in this course).
Types of Infinite Horizon Problems

1. Total cost problems

\[
\min_{\{\mu_k\}} \lim_{N \to \infty} \mathbb{E} \left[ \sum_{k=0}^{N-1} g(x_k, \mu(x_k), w_k) \right]
\]

- Not commonly used because cost function often goes to infinity.
- Special type called stochastic shortest path problem with cost free termination state studied in Bertsekas.

2. Discounted cost problems

\[
\min_{\{\mu_k\}} \lim_{N \to \infty} \mathbb{E} \left[ \sum_{k=0}^{N-1} \gamma^k g(x_k, \mu_k(x_k), w_k) \right]
\]

where \( \gamma \in (0, 1) \).

- \( \gamma \) is called the discount factor.
- Cost incurred at earlier times more important than later times.
Types of Infinite Horizon Problems

3 Average cost problems

\[
\min_{\{\mu_k\}} \lim_{N \to \infty} \mathbb{E} \left[ \frac{1}{N} \sum_{k=0}^{N-1} g(x_k, \mu(x_k), w_k) \right]
\]

- Cost incurred in the future more important than at the beginning
- Cost function usually finite in contrast to total cost problems.
- Optimal average cost usually independent of initial states.
Discounted Cost Problems

\[
\min_{\{\mu_k\}} \lim_{N \to \infty} \mathbb{E} \left[ \sum_{k=0}^{N-1} \gamma^k g(x_k, \mu_k(x_k), w_k) \right]
\]

Assume

- number of states is finite, taking values 1, 2, ..., \(n\).
- number of possible controls is finite.
Notation

- Given policy $\pi = \{\mu_0, \mu_1, \ldots, \}$, cost of policy starting at state $i$ is

$$J_\pi(i) = \lim_{N \to \infty} \mathbb{E}\left[ \sum_{k=0}^{N-1} \gamma^k g(x_k, \mu_k(x_k), w_k) \middle| x_0 = i \right]$$

- If policy is stationary, i.e. $\pi = \{\mu, \mu, \ldots\}$, write $J_\mu(i)$ instead of $J_\pi(i)$.

- Optimal cost starting at state $i$ is

$$J^*(i) = \min_{\pi} \lim_{N \to \infty} \mathbb{E}\left[ \sum_{k=0}^{N-1} \gamma^k g(x_k, \mu_k(x_k), w_k) \middle| x_0 = i \right]$$

- Optimal policy $\pi^*$ satisfies $J_{\pi^*}(i) = J^*(i)$, $\forall i$. 
Theorem

For the discounted cost problem we have:

(a) The value iteration algorithm

\[ J_{k+1}(i) = \min_{u \in U(i)} \mathbb{E}[g(i, u, w) + \gamma J_k(f(i, u, w))], \quad i = 1, 2, \ldots, n \]

converges as \( k \to \infty \) to optimal costs \( J^*(i) \), \( i = 1, 2, \ldots, n \), starting from arbitrary \( J_0(i) \), \( i = 1, 2, \ldots, n \)

(b) The optimal costs \( J^*(i) \), \( i = 1, 2, \ldots, n \), satisfy the Bellman equation

\[ J^*(i) = \min_{u \in U(i)} \mathbb{E}[g(i, u, w) + \gamma J^*(f(i, u, w))], \quad i = 1, 2, \ldots, n \]
Theorem
(c) Given a stationary policy \( \mu \), the cost \( J_\mu(i), i = 1, \ldots, n \) satisfies

\[
J_\mu(i) = \mathbb{E}[g(i, \mu(i), w) + \gamma J_\mu(f(i, \mu(i), w))], i = 1, \ldots, n
\]

Starting from arbitrary \( J_0(i), i = 1, \ldots, n \), the iteration

\[
J_{k+1}(i) = \mathbb{E}[g(i, \mu(i), w) + \gamma J_k(f(i, \mu(i), w))]
\]

converges to \( J_\mu(i), i = 1, \ldots, n \)

(d) A stationary policy \( \mu \) is optimal iff. for every state \( i, \mu(i) \) attains the minimum in the Bellman equation.

(e) The policy iteration algorithm

\[
\mu_{k+1}(i) = \arg\min_{u \in U(i)} \mathbb{E}[g(i, u, w) + \gamma J_{\mu_k}(f(i, u, w))], i = 1, 2, \ldots, n
\]

generates an improving sequence of policies and terminates (in finite time) with an optimal policy.

Proof: see Bertsekas
Parts (a) and (e) provide algorithms for solving discounted cost problems (like the D.P. algorithm for finite horizon problems). Value iteration (part (a)) requires less computation at every iteration, while policy iteration (part (e)) is guaranteed to terminate in finite time.

In part (c),

\[ J_\mu(i) = \mathbb{E}[g(i, \mu(i), w) + \gamma J_\mu(f(i, u, w))], \ i = 1, \ldots, n \]

is a system of \( n \) linear equations, with which one can solve for \( J_\mu(i), \ i = 1, \ldots, n \).
Policy Iteration

- Starting with a stationary policy $\mu_0$, generate a sequence $\mu_1, \mu_2, \ldots$ of stationary policies.

- Given $\mu_k$, perform policy evaluation step, to compute $J_{\mu_k}(i), i = 1, 2, \ldots, n$, using
  \[ J_{\mu_k}(i) = \mathbb{E}[g(i, \mu_k(i), w) + \gamma J_{\mu_k}(f(i, \mu_k(i), w))], \quad i = 1, \ldots, n \]

- Given $J_{\mu_k}(.),$ perform policy improvement step
  \[ \mu_{k+1}(i) = \text{argmin}_{u \in U(i)} \mathbb{E}[g(i, u, w) + \gamma J_{\mu_k}(f(i, u, w))] \]
  with $J_{\mu_k}(f(i, u, w))$ the “cost to go of old policy” (c.f. rollout algorithm)

- Terminate when $J_{\mu_k}(i) = J_{\mu_{k+1}}(i), \forall i.$
Example

A manufacturer at each time period:
- Receives an order with probability $p$, no order with probability $1 - p$.
- May process all unfilled orders at cost $K > 0$, or process no orders.
- Cost per unfilled order at each time period is $C > 0$.
- max. no. unfilled orders is $n$.

Find processing policy that minimizes the discounted cost, with discount factor $\gamma$. 
Example

- Let state = no. unfilled orders at the start of each period \((\in \{0, 1, \ldots, n\})\).
- Bellman equation:
  For states \(i = 0, 1, \ldots, n - 1\), can either process orders or not, so Bellman equation is

\[
J^*(i) = \min \left\{ \begin{array}{c}
K_{\text{process}} + \gamma p \ J^*(1) + \gamma (1 - p) \ J^*(0), \\
C_{\text{don't process}} + \gamma p \ J^*(i + 1) + \gamma (1 - p) \ J^*(i)
\end{array} \right\}
\]

For state \(i = n\), all orders must be processed, so Bellman equation is

\[
J^*(n) = K + \gamma p J^*(1) + \gamma (1 - p) J^*(0)
\]

- Can show that the optimal policy is a threshold policy: process order iff. \(i \geq m^*\), where \(m^*\) is a threshold (see Bertsekas).
Average Cost Problems

\[
\min_{\{\mu_k\}} \lim_{N \to \infty} \frac{1}{N} \mathbb{E} \left[ \sum_{k=0}^{N-1} g(x_k, \mu_k(x_k), w_k) \right]
\]

- In most problems of this type, the average cost per stage of a policy is independent of initial state.
- Expresses costs occurred in the long run, costs incurred in early stages do not matter.
- Analysis is harder than for discounted cost problems (won’t cover here).
- Assume
  - number of states is finite, taking values 1, 2, ..., \(n\).
  - number of possible controls is finite.
- Also assume that there is some state \(t\) such that for all initial states and policies, \(t\) is visited infinitely often with probability 1.
Theorem

For the average cost problem, we have:

(a) The optimal average cost per stage $\lambda^*$ is the same for all initial states, and there exists a vector $h^* = (h^*(1), h^*(2), ..., h^*(n))$ satisfying the Bellman equation:

$$\lambda^* + h^*(i) = \min_{u \in U(i)} \mathbb{E}[g(i, u, w) + h^*(f(i, u, w))], \quad i = 1, \ldots, n$$

($h^*$ is unique if we fix $h^*(t) = 0$.)

If $\mu(i)$ attains the minimum in the Bellman equation for all $i$, then the stationary policy $\mu$ is optimal.

(b) If $\mu$ and $h$ satisfy Bellman’s equation, then $\lambda$ is the optimal average cost per stage for each initial state.
Theorem

(c) Given a stationary policy $\mu$ with average cost per stage $\lambda_\mu$, there exists a vector $h_\mu = (h_\mu(1), ..., h_\mu(n))$ such that

$$\lambda_\mu + h_\mu(i) = \mathbb{E}[g(i, \mu(i), w) + h_\mu(f(i, \mu(i), w))], \ i = 1, ..., n$$

($h_\mu$ is unique if we fix $h_\mu(t) = 0$.)

Proof: See Bertsekas

Comment: $h$ is also called the differential cost vector.
A manufacturer at each time period:

- Receives an order with probability $p$, no order with probability $1 - p$.
- May process all unfilled orders at cost $K > 0$, or process no orders.
- Cost per unfilled order at each time period is $C > 0$.
- max. no. unfilled orders is $n$.

Find processing policy that minimizes the average cost.
Example

- State = no. unfilled orders at start of each period.
  State 0 = Special state $t$ here (will visit this state infinitely often)
- Bellman equation:
  For states $0, 1, \ldots, n - 1$, Bellman equation is
  \[
  \lambda^* + h^*(i) = \min \{ K + ph^*(1) + (1 - p)h^*(0), C_i + ph^*(i + 1) + (1 - p)h^*(i) \}
  \]
  For state $n$, Bellman equation is
  \[
  \lambda^* + h^*(n) = K + ph^*(1) + (1 - p)h^*(0)
  \]
- Optimal policy: Process orders if
  \[
  K + ph^*(1) + (1 - p)h^*(0) \leq C_i + ph^*(i + 1) + (1 - p)h^*(i)
  \]
- Can again show that a threshold policy is optimal, where value of the threshold may be different from value of the threshold in discounted cost problem
Value iteration:

- Starting from any \( J_0 \), compute

\[
J_{k+1}(i) = \min_{u \in U(i)} \mathbb{E}[g(i, u, w) + J_k(f(i, u, w))], \; i = 1, \ldots, n
\]

- Have

\[
\lim_{k \to \infty} \frac{J_k(i)}{k} = \lambda^*, \; \forall i
\]

Drawbacks of value iteration:

- Often components of \( J_k \) will diverge to \( \infty \) or \( -\infty \), so calculating

\[
\lim_{k \to \infty} \frac{J_k(i)}{k}
\]

may be tricky.

- Doesn’t compute a differential cost vector \( h^* \).
Relative value iteration:

- Subtract a constant (dependent on \(k\)) from all components of \(J_k\), so that the difference \(h_k\) is bounded, e.g.

\[
h_k(i) = J_k(i) - J_k(s), \quad i = 1, ..., n
\]

where \(s\) is some fixed state. Then relative value iteration algorithm is:

\[
h_{k+1}(i) = \min_{u \in U(i)} \mathbb{E}[g(i, u, w) + h_k(f(i, u, w))] - \min_{u \in U(s)} \mathbb{E}[g(s, u, w) + h_k(f(s, u, w))], \quad i = 1, ..., n
\]

- Can show that \(h_k \to h^*\) as \(k \to \infty\)
Algorithms For Average Cost Problems

Policy iteration:

- Given $\mu_k$, perform **policy evaluation step** to compute $\lambda_k$ and $h_k$, using the equations:

  $$\lambda_k + h_k(i) = \mathbb{E}[g(i, \mu_k(i), w) + h_k(f(i, \mu_k(i), w))], \quad i = 1, \ldots, n$$

  $$h_k(t) = 0$$ for some state $t$ which is visited infinitely often.

- Given $\lambda_k$ and $h_k$, perform **policy improvement step**:

  $$\mu_{k+1}(i) = \arg\min_{u \in U(i)} \mathbb{E}[g(i, u, w) + h_k(f(i, u, w))], \quad i = 1, \ldots, n$$

- Terminate when $\lambda_{k+1} = \lambda_k$ and $h_{k+1}(i) = h_k(i), \quad i = 1, \ldots, n$.

  Policy iteration can be shown to terminate in finite time.
Introduction to Reinforcement Learning
Introduction to Reinforcement Learning

- As in the finite horizon case, want to consider suboptimal methods for solving infinite horizon problems
- Also studied in machine learning as reinforcement learning
- Many different methods, e.g. Q-learning, TD/SARSA(\(\lambda\)), REINFORCE, . . .
Introduction to Reinforcement Learning

Slight change of notation:

- State $x_k \rightarrow$ state $s_k$
- Control $u_k \rightarrow$ action $a_k$
- Cost function $g(.,.,.) \rightarrow$ Reward function $g(.,.,.)$
- Cost minimization $\min \lim_{N \rightarrow \infty} \mathbb{E} \left[ \sum_{k=0}^{N-1} \gamma^k g(x_k, \mu_k(x_k), w_k) \right]$
- Reward maximization $\max \lim_{N \rightarrow \infty} \mathbb{E} \left[ \sum_{k=0}^{N-1} \gamma^k g(s_k, a_k(s_k), w_k) \right]$
Q-Learning for Discounted Problems

- Bellman equation

\[ J^*(s) = \max_a \mathbb{E}[g(s, a, w) + \gamma J^*(f(s, a, w))], \forall s \]

\( J^*(s) \) is the optimal expected future reward when in state \( s \)

- Introduce now the Q-Bellman equation

\[ Q^*(s, a) = \mathbb{E}[g(s, a, w) + \gamma \max_{a'} Q^*(s', a')], \forall (s, a) \]

where \( s' \triangleq f(s, a, w) \).

- Q-factor \( Q(s, a) \) is the expected future reward when in state \( s \) and taking action \( a \)

- \( Q^* \) are the optimal Q-factors
Q-Learning for Discounted Problems

- Can also solve Q-Bellman equation using value iteration or policy iteration
- Given $Q^*(s, a)$, optimal policy can be computed as
  \[ a^*(s) = \arg\max_a Q^*(s, a) \]
- Using $Q^*(s, a)$ gives same policy as using $J^*(s)$, though it requires more storage
- However, one advantage is that $Q^*(s, a)$ can be found approximately using e.g. $Q$-learning
Q-Learning for Discounted Problems

Q-learning algorithm. Repeat:

- Generate \((s_k, a_k)\) using any probabilistic mechanism such that all state-action pairs \((s, a)\) are chosen infinitely often
- Given \((s_k, a_k)\), update \(Q(s_k, a_k)\) as:

\[
Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left( r + \gamma \max_{a'} Q_k(s', a') - Q_k(s_k, a_k) \right)
\]

where \(r = g(s_k, a_k, w_k)\) is the sampled reward, \(s' = f(s_k, a_k, w_k)\) is the sampled next state when current state is \(s_k\) and action \(a_k\) is applied, and \(\{\alpha_k\}\) is a sequence converging to 0.

- Leave all other Q-factors unchanged
Q-Learning for Discounted Problems

- Q-learning algorithm converges to the optimal Q-factors \( Q^*(s, a) \) provided all pairs \((s, a)\) are chosen infinitely often, and the sequence \(\{\alpha_k\}\) satisfies

  \[
  \alpha_k > 0, \quad \sum_{k=0}^{\infty} \alpha_k = \infty, \quad \sum_{k=0}^{\infty} \alpha_k^2 < \infty
  \]

  - e.g. \( \alpha_k = \frac{1}{k} \) satisfies this condition

- In Sutton & Barto, \(\{(s_k, a_k)\}\) generated according to:

  \[
  s_{k+1} := s' \\
  a_{k+1} = \begin{cases} 
  \text{random } a, & \text{w.p. } \epsilon \\
  \arg \max_a Q_{k+1}(s_{k+1}, a), & \text{w.p. } 1 - \epsilon,
  \end{cases}
  \]

  for some \( \epsilon > 0 \)
Function Approximation and Deep Reinforcement Learning

For large problems:
- Too many (state, action) pairs to store in memory
- Too slow to learn the value of each $Q^*(s, a)$ individually

Function approximation:
- Regard $Q^*(s, a)$ as a function of $s$ and $a$. Approximate $Q^*(s, a)$ by another function $\hat{Q}(s, a, \theta)$ parameterized by a set of weights $\theta$
- Learn the weights $\theta$ instead of the entire set of values $Q^*(s, a)$

Deep reinforcement learning: When the weights $\theta$ are learnt using a deep neural network, see https://deepmind.com/blog/deep-reinforcement-learning

Spectacular recent advances in AI using deep reinforcement learning, e.g. AlphaGo, AlphaZero
Deep Reinforcement Learning - Further Reading

- Overview
  - https://deepmind.com/blog/deep-reinforcement-learning
  - http://www0.cs.ucl.ac.uk/staff/d.silver/web/Talks.html

- Deep Q-Network (DQN) algorithm
  - https://keon.io/deep-q-learning

- More Advanced
  - https://medium.com/tensorflow/deep-reinforcement-learning-playing-cartpole-through-asynchronous-advantage-actor-critic-a3c-7eab2eea5296