史丰源

BERTSEKAS 2014 年在清华的 *ADP* 课程笔记

Contents

Part I

INTRODUCTION

1

Introduction

CONTENTS

这个 ADP 课程[3] 一共有六天，每天一个 lecture，课程的主要是为了做研究的人讲的，所以会比较前沿，有很多理论与应用。因为 ADP 内容比较多，这六个课程会比较简略地做介绍，如果想更深入地了解相关内容，可以去看这三本书："Neuro-Dynamic Programming"，"Dynamic Programming and Optimal Control, Vol. II: Approximate Dynamic Programming" 和 "Abstract Dynamic Programming" [1] [2] [5]，这个课程主要讲的也是这三本书的内容，如果还想要了解更详细的信息，可以访问 http://web.mit.edu/dimitrib/www/publ.html 主页下载文献和书之类的资料。

1.1 BRIEF OUTLINE I

这个课程的主题是基于近似 (一部分基于仿真) 的大规模动态规划，这是一个近几年受到很多关注的领域，不同背景的研究者给这个领域起了不同的名字，人工智能或者机器学习的研究者叫它强化学习，我一般叫它神经动态规划或者近似动态规划，目前主要有两个方向，某种程度上他们是互相融合的，一个方向是人工智能，也就是强化学习，主要依赖于基于特征表达的输入输出之间的函数关系，也可以叫基于观察样本或者仿真进行学习，另一个方向是最优控制理论，最优控制理论的关注重点是传统的优化方法和算法，比如策略迭代和值迭代什么的。这两个方向的研究者认识到他们正在从两个不同的方向出发解决同一个类型的问题，这两个方向的研究获得了很多成果。

我们要研究的是用动态规划求解不确定性的控制问题，这个不确定性，不仅仅指随机，还有可能是解决 $\min\max$ 的问题，这个时候不确定性就可以来源于对手做出的行为了 (还有其他来源，对手的行为只是其中一种)，比如游戏或者多阶段游戏，动态规划在这类问题中应用得非常广泛。实际上动态规划可以求解动态系统的控制问题，比如离散优化问题和组合最优化问题。

大家都认同动态规划的应用非常广泛，但是动态规划并不能直接有效地解决实际应用规模下的问题，想要求解实际应用规模的问题，需要在动态规划

的基础上再做一些工作，现在已经有很多工作让动态规划能够求解所有场景下展望期非常长的大规模问题了，比如控制理论，运筹优化，经济问题，人工智能和金融问题等。

动态规划是一种一般性的方法，有很多理论支撑，有很多很有挑战的问题待解决，如何设计合适的方法解决这些问题很重要，还有很多抽象的理论问题需要研究，比如如何建立一个合适的动态规划模型，这个课程的时间不是很多，所以不会涉及这些内容，会把主要的精力放在算法上。

1.2 BRIEF OUTLINE II

这个课程的目标是展示如何使用近似或者仿真技巧求解问题，需要解决的是动态规划的两个问题，bellman 提出动态规划时提出了“维数灾”的概念，也就是使用动态规划求解大规模问题时会遇到困难，也是动态规划的主要局限性，使用动态规划的另一个困难是系统模型不可知。想要在一个具体的问题上使用动态规划，就需要这个系统的数学模型，比如状态转移方程之类的东西，但是有系统没法得到数学模型，就只能使用仿真来代替数学模型了。举个例子，一个实时的仿真系统，无法用数学模型来求解，就只能用模拟器代替数学模型然后求解。

这个课程的安排是这样的，前两次课程讲精确动态规划，包括有限期和无限期问题，其中无穷维问题是课程关注的重点，第三次课程会讲一些算法和求解大规模问题的方法，在这里会引入近似和仿真技巧，接下来会将一些特殊的动态规划方法，比如基于策略方向的方法，时域差分，还会用一次课程的时间介绍时域差分的不同算法，比如 q-learning 和策略空间近似等方法。总的来说，前两次课程是后面四个课程的基础。

Part II

EXACT DYNAMIC PROGRAMMING: OVERVIEW

2

LECTURE 1: BASIC CONCEPT OF DP

CONTENTS

2.1 LECTURE OUTLINE

这个课程要介绍的是经典的有限期动态规划算法，然后把算法扩展到无限期问题上去，并且开发一些能解决这些无限期问题，比如阶段平均成本有界的离散问题的基础理论，主要资料是教授之前写过的三本书。

这个课程主要包括以下内容：

1. 介绍动态规划和近似动态规划
2. 有限期问题
3. 有限期问题的动态规划算法

4. 无限期问题
5. 折扣无限期问题的基础理论

2.2 DP AS AN OPTIMIZATION METHODOLOGY

在研究优化问题的时候，你可以发现一件事请，很多领域的很多问题都会产生很多不同形式的优化问题，如果从更抽象的角度来看，只有一种优化问题，就是最小化某一个目标 (最大化问题加一个负号也可以变成最小化问题)，也可以叫最小化问题，有时候需要优化一个标量，有时候要优化一个向量，或者更复杂的优化目标。优化的时候要受到约束集合的限制，也就是决策 u 是收到限制的，优化就是从约束集合给出的决策集合 U 中选择合适的 u 让目标值最小。

现在有一个函数 $g(u)$，需要解决的问题是 $\min_{u\in U} g(u)$，即从决策空间中找到一个合适的 u 让 $g(u)$ 的值最小。这个问题有两种类型，一个是从有限集，即离散空间中寻找最优的 u，另一个是从无限集合，即连续空间中寻找最优的 u，当 u 是连续值时，可以取的值的数量是无限多个，我们把从离散空间中找最优决策的问题叫组合问题，把从连续空间中找最优决策的问题叫连续问题，他们是不一样的，也是优化问题的一个主要的分类标准。计算机科学家比较喜欢解决整数规划问题，整数规划问题有与连续问题 (微积分相关方法或者凸优化相关方法) 不同的特征，这也是离散问题与连续问题最大的不同点。

另一个分类方法是线性与非线性，线性指的是目标函数与约束都是线性的，也就是目标函数是线性函数，决策变量定义在多边形集上 (或多边形集内)。非线性问题指的是目标函数与约束都是非线性的。

第三个基本的分类方法也是主要的分类方法是把问题分为随机问题 (其实这里应该叫不确定性问题) 与确定性问题。不确定性问题包括一个随机参数 w，求期望的时候可以对 w 求平均或者对 w 相关的表达式求平均。给定控制 u 与随机变量 w, 可以改写成本函数 $g(u) = E_w\{G(u, w)\}$，计算 $G(u, w)$ 对分布 w 下的期望得到目标函数 g 的值，然后就可以在 u 的定义域内搜索最优解了，从这个角度上讲，不确定性问题与确定性问题没有什么区别，唯一的一个区别就是不确定性问题有一个随机量，这个随机量会对选择求解问题的方法的时候产生很大的影响。

上面介绍了三种标准对优化问题进行分类，一共有三种，分别是：

1. 连续问题与离散问题
2. 线性问题与非线性问题
3. 不确定性问题与确定性问题

动态规划算法可以同时求解线性，非线性，不确定性，确定性，离散与连续问题，但是从动态规划的角度上来说，对于问题的分类主要的标准是是不是多阶段问题。一个多阶段问题需要决策很多次，在某一阶段决策选择一个 u，然后执行这个控制 u，观测到执行这个控制后的信息时再选择一个新的控

制，然后继续进行“决策-执行-决策”的过程。这就是动态规划能求解的问题的特点，随机信息 w 会在产生新阶段的时候产生，上一次决策对新阶段的状态产生影响，然后利用观测的新状态进行决策。由于动态规划能求解的问题需要满足某些特性 (无后效性或者马尔科夫性)，所以一个整数规划能求解的问题，可能就不能用动态规划求解，另一个很大的区别是动态规划能得到全局最优解 (bellman 最优性能够保证最优解存在)，而其他一部分方法难以得到全局最优解，只能得到局部最优解。

2.3 BASIC STRUCTURE OF STOCHASTIC DP

有一个离散时间动态系统 $x_{k+1} = f_k(x_k, u_k, w_k), k = 0, 1, \cdots, N-1$，系统中 k 为时间索引，取值范围为 0 到 N，系统状态 x_k 表示时间为 k 时的系统的状态，时间 N 时出现的系统状态 x_N 叫做终止状态，w_k 是一个随机参数，一些人叫它噪声，也有叫概率分布的，主要依赖于上下文，N 是决策的展望期，或者时间的数量，即需要决策的次数或者控制被执行的次数。我们需要做的就是观察到系统状态 x_k 的时候做出最好的决策 u_k，而且不需要知道系统是如何到达当前状态的，只需要知道现在系统的状态是什么。

考虑一个有限期问题，系统从状态 x_0 开始，然后选择控制 u_0 并执行，执行控制时受到随机变量 w_0 的影响产生新的系统状态 x_1，然后选择控制 u_1，执行，观察新状态，不停地重复这个操作直到时间结束。每次执行控制 u_k 并受到 w_k 干扰都会产生成本 g_k，随着控制不断被执行，把所有成本 g_k 累加起来，终止状态 x_N 产生的时候成本累加的结果就是这个系统从初始状态在这一系列控制下产生的总成本，由于终止状态会有一个与控制无关的成本，所以形式是 $E\left\{g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)\right\}$，这个函数也是成本函数的定义，同时由于函数值受到随机因素的影响，一般使用累加成本的期望值作成本函数，优化时的目标也是这个期望总成本最低。

系统的状态方程 $x_{k+1} = f_k(x_k, u_k, w_k), k = 0, 1, \cdots, N-1$ 有另一种表达方式，$P(x_{k+1}|x_k, u_k)$ 是一个概率分布，表示当前系统状态 x_k 下执行控制 u_k 产生新状态 x_{k+1} 的概率分布，这样你就可以非常简洁地使用 $P(x_{k+1}|x_k, u_k)$ 来描述这个系统。举一个特殊的例子，状态 $x_{k+1} = w_k$，这种情况非常特殊了，会有效率更高的方法求解。这种系统描述方法是通过给定转移概率来描述系统，同时适用于不确定性系统与确定性系统，不确定性系统可以给出概率，确定性系统的概率是 1，这种形式也就同时用于确定性与不确定性系统描述了。这两种方法是等价的，都可以用来描述一个系统，所以在描述一个系统或者看到别人用任意一种方法时，你都要知道这个系统是什么样子的。

实际上这两种方法有一点缺陷，或者说描述系统的方法有一点缺陷，状态转移函数和状态转移概率这两种方法，状态转移函数的缺陷是有些系统可能非常难以得到解析表达的函数形式，状态转移概率的缺陷是如果使用数值概率，也就是给出一个状态转移概率矩阵的话，实例中状态和控制会非常多 (离散状态和离散控制)，矩阵空间非常大，甚至会导致计算机内存溢出，如果状态和控制有一个是连续的，数值概率矩阵会有无数个维度；如果使用解析的

概率分布函数，无论状态和控制是连续还是离散都可以表达，但是这个解析关系非常难得到，所以 DP 能研究的就只有状态转移能够得到的问题，包括：数值状态转移概率矩阵、状态转移概率分布函数、状态转移函数。这三个元素任意一个能够得到都可以直接求解，如果这三个元素全都无法获得，就可以使用仿真技术进行采样，然后求解。

2.4 INVENTORY CONTROL EXAMPLE

举一个库存控制的例子，仓库中有一些货物，然后来了一批客户，这时决策者需要把货物卖给客户满足他们的需求，然后下订单补充货物。在这个库存系统中，x_k 是阶段 k 时的库存状态，比如一个存放汽车的仓库，放了 x_k 数量的汽车，来了几个客户，对汽车的需求是 w_k，如果完全满足了他们的需求，库存量减少为 $x_k - w_k$，下了订货量为 u_k 的订单补充库存，订单到达后 (假设订单到达前没有卖出汽车给客户) 新的库存状态为 $x_k - w_k + u_k$。在下订单时需要付出订货成本，已知确定的汽车价钱 c，也就是 u_k 的单价，则需要支付的订货成本是 cu_k，如果上阶段结束后仓库中还留有一部分库存，那么经过一个阶段，需要支付库存成本，库存不能太多，否则会产生大量的库存成本，也不能太少，否则无法满足新阶段的客户需求，综合这两种情况，库存成本就包括货物过多的库存成本或货物过少的缺货成本，而终止状态 x_N 会产生剩余库存成本 $g_N(x_N)$，这就是一个离散时间系统，状态方程为 $x_{k+1} = f_k(x_k, u_k, w_k) = x_k - w_k + u_k$，一个经典的动态规划问题，如果剩余库存不要了，全都丢掉的话你可以令 $g_N = 0$，这样这个系统会更好分析，可以看到令 $g_N = 0$ 后的系统总成本变为 $E\left\{g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)\right\} = E\left\{\sum_{k=0}^{N-1} (cu_k + r(x_k + u_k - w_k))\right\}$。在这个系统中，$w_k$ 是随机变量，u_k 必须满足库存平衡约束，即不能让库存小于零，订单必须让库存保持在大于或等于 0 的水平，有些系统约束会依赖于系统状态，比如仓库容量上限是 100 辆车，那么不管什么情况订单都不可以超过 100 量汽车，每阶段订单不能超过剩余库存空间。

2.5 ADDITIONAL ASSUMPTIONS

这个 (库存) 模型还有一些假设，首先是随机量 w_k 不依赖于过去的信息，也就是说 w_k 只有可能依赖于上一阶段的 x_k 和 u_k，也可能不依赖，而且 w_k 与之前的需求不相关，也不会受到之前的需求的影响。如果需求依赖于过去的信息，那么过去的信息也能够为优化问题提供部分可以利用的信息，这会让问题变得更困难。同时每阶段的控制收到当前状态 x_k 的限制而不受到之前的状态或者控制的限制。这两个假设共同构成了马尔科夫性，或者叫无后效性，即当前决策只受到当前状态影响，下一状态也只受上一个状态与控制影响，而不受到更早的信息的影响，如果实际系统中不满足这个条件，就需要建立一

个满足这个性质的模型，现在假设一个已有的系统随机变量不依赖于过去的信息，新状态也只受到上一个状态与控制的影响，每一阶段的控制都只受到当前状态的约束。

下面要讲的两件事不是假设，而是求解思路。首先是控制策略 $u_k = \mu_k(x_k)$，是一个从系统状态到控制的映射，这个映射在控制领域通常被叫做策略，作用是告诉我们如何在当前状态下进行控制，上面的库存问题就是给出当前库存下如何订货，这是一种反馈控制律，能够给出控制后观察到控制执行的结果，一般是一个函数。

在优化的时候，我们优化的并不是控制，而是控制律，也就是策略函数，现在有一个 N 期的问题，需要决策 N 次，那么我们要优化的就是 N 个控制策略函数，而不是 N 个控制的值。这种做法的好处就是优化结果是控制策略的向量而不是控制向量，这样在系统状态发生改变的时候还可以直接使用优化好的控制律进行控制，而不需要像优化控制向量那样重新计算最优的控制。另一个好处是直接在所有期的控制向量空间中搜索最优的控制向量需要搜索的空间非常巨大，比搜索每一期的控制律空间要大得多，所以线性规划和非线性规划用来求解这些问题时很容易遇到问题，而反馈控制律能求解并且有比较好的表现。

2.6 GENERIC FINITE-HORIZON PROBLEM

下面正式详细描述一下一般的有限期动态系统，状态方程 $x_{k+1} = f_k(x_k, u_k, w_k)$ 在给定了 x_k，u_k 和 w_k 之后会转移到新状态 x_{k+1}，控制 $u_k \in U_k(x_k)$ 的取值范围受到当前状态的约束，依赖于系统状态 x_k 和控制 u_k 的随机变量 w_k 的条件分布概率已知，所以我们也可以得到状态 x_k 和控制 u_k 下被 w_k 影响产生新状态的概率分布 $P_k(\cdot|x_k, u_k)$。使用控制策略 π，即从状态到控制的映射序列 $\{\mu_0, \mu_1, \cdots, \mu_{N-1}\}$ 进行控制。策略 π 作用下的期望总成本被定义为：$J_\pi(x_0) = E\left\{g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k)\right\}$ 这个期望总成本是策略与初始状态相关的，即对于给定的策略和初始状态算出的期望总成本，一般被称为策略的成本函数，给定一个策略，每一个初始状态都可以算出相应的期望总成本。优化目标是通过寻找最优策略最小化期望总成本，给出两个定义：$J^*(x_0) = \min_\pi J_\pi(x_0)$ 和 $J_{\pi^*}(x_0) = J^*(x_0)$，$J^*(x_0)$ 是在整个策略空间中寻找最优策略得到的最优成本，对于给定的相同的初始状态，不同的策略会得到不同的策略成本，选择策略成本最小的那个策略，就是整个问题的最小成本，也就是 $J_{\pi^*}(x_0) = J^*(x_0)$ 表示的这样。

这里需要重点强调一下，最优成本对策略和初始状态的依赖，最优成本不会依赖于初始状态但是依赖于策略，也就是说，选定了一个最优策略，无论初始状态是什么，得到的都是该初始状态下的最优成本，该策略也是这个初始状态下的最优策略，所以在寻找最优策略时需要遵守的一个很重要的前提是，搜索在所有初始状态下策略成本都最小的那个策略，有些时候“所有初始状态”没法严格保证，所以会产生一些妥协，有时候是“可能出现的初始状态”，

有时候可以是对所有状态策略成本的加权最小，这就需要具体问题具体分析了。

这就是一般性的有限期问题的数学描述，需要注意的是这个描述没有对状态和控制进行任何假设，也就是说对于任意的状态都可以算出相应的成本。

举几个例子，飞机在天上飞，这个飞机的运动由六个状态值定义，位置的三个坐标和速度的三个坐标共同构成了一个六维的状态向量，而且状态向量的每一个维度都从连续实数空间内取值。另一个例子，状态在有限离散空间内取值，比如排队系统，状态一般由队列中客户的数量构成，假设队列中客户数量是有限的，那么整个的状态空间就是有限的，如果排队系统从队列变成了排队网络，那么状态空间从原来的队列状态空间进行笛卡儿积得到，还是一个有限集合。

动态规划的好处就是，不依赖于状态和控制的空间，连续空间和离散空间都可以使用相同的算法，只是需要根据问题的特性进行修改，这种性质很重要，它可以让这种技术具有通用性，简单地在欧几里得空间、有限集合，比如开关决策问题中应用。

2.7 PRINCIPLE OF OPTIMALITY

使用动态规划求解一个问题得到了最优策略，这个最优策略是不是全局最优解是需要证明的，也就是动态规划最核心的东西，bellman 最优性原理，同时它也是一个很不重要的东西，因为最优性原理不需要对所有的问题都做证明，只要你能构造一个问题的动态规划模型，动态规划方法求得的就是这个问题的全局最优解。

现在我们要求解一个 N 期的最优策略，需要搜索一个 N 个函数的策略，最优策略记作 $\pi^* = \left\{\mu_0^*, \mu_1^*, \cdots, \mu_{N-1}^*\right\}$，从第 0 期开始，到第 N 期结束，这个策略的 N 个映射共同作用得到最优成本。如果要求解的是从第 k 期开始，直到第 N 期结束的最优策略序列，得到的是整个问题的最优策略的一部分，这个从 x_k 开始的问题叫做原问题的子问题，子问题的控制同样受到原问题的约束，假设我找到了整个问题的最优策略，还定义了子问题的表达式，在时间 k 观测到系统状态 x_k，从第 k 期到第 N 期的累加成本 $E\left\{g_N(x_N) + \sum_{l=k}^{N-1} g_l(x_l, \mu_l(x_l), w_l)\right\}$ 被叫做"cost-to-go"，由于子问题规模比原问题小，因此子问题更好解决，子问题的最优策略也是原问题最优策略从时间 k 到时间 N 对应的策略。

子问题的最优策略是原问题最优策略的一部分的原因是，从倒数第二个状态 x_{N-2} 出发，决策一次到达终止状态 x_N，这时最优策略 μ_{N-1}^* 是一个映射，然后把时间向前移动一步，从倒数第三个状态出发 x_{N-2}，决策两次到达终止状态 x_N，搜索最优策略即最小化

$$
\begin{aligned}
&E\left\{g_{N-2}(x_{N-2}, \mu_{N-2}(x_{N-2}), w_{N-2}) + g_{N-1}(x_{N-1}, \mu_{N-1}(x_{N-1}), w_{N-1}) + g_N(x_N)\right\} \\
=&E\left\{g_{N-2}(x_{N-2}, \mu_{N-2}(x_{N-2}), w_{N-2}) + J_{\mu_{N-1}^*}(f_k(x_{N-2}, u_{N-2}, w_{N-2}))\right\}
\end{aligned}
$$

所以每次求解时需要用到下一阶段直到最后一阶段的最优策略，这个公式中间跳了一步，因为某子问题求最优解的形式是

$$\min_{\mu_{N-2},\mu_{N-1}} E\left\{g_{N-2}(x_{N-2},\mu_{N-2}(x_{N-2}),w_{N-2})+g_{N-1}(x_{N-1},\mu_{N-1}(x_{N-1}),w_{N-1})+g_N(x_N)\right\}$$

$$\Rightarrow \min_{\mu_{N-2},\mu_{N-1}} E\left\{g_{N-2}(x_{N-2},\mu_{N-2}(x_{N-2}),w_{N-2})+J_{\mu_{N-1}}(f_k(x_{N-2},u_{N-2},w_{N-2}))\right\}$$

$$\Rightarrow \min_{\mu_{N-2}} E\left\{g_{N-2}(x_{N-2},\mu_{N-2}(x_{N-2}),w_{N-2})+J_{\mu^*_{N-1}}(f_k(x_{N-2},u_{N-2},w_{N-2}))\right\}$$

上述变化中，第一次变换是等价的，同时 $J_{\mu^*_{N-1}}(f_k(x_{N-2},u_{N-2},w_{N-2}))\le J_{\mu_{N-1}}(f_k(x_{N-2},u_{N-2},w_{N-2}))$ 对于所有 μ_{N-1} 都严格成立，所以第二个变换也是等价的，因此当前子问题的最优策略序列和该子问题的子问题的最优策略序列是真包含于的关系。

视频中还举了个例子，笔者不想详细举例子说明了，如果还是不清楚的话，可以用一个最短路径的算例自己完成一遍逆序过程就可以了。

2.8 DP ALGORITHM

首先定义状态 x_k 的成本函数

$$J_k(x_k)=\min_{u_k\in U_k(x_k)} E\left\{g_k(x_k,u_k(x_k),w_k)+J_{k+1}(f_k(x_k,u_k,w_k)\right\}$$

$$u_k^*=\arg\min_{u_k\in U_k(x_k)} E\left\{g_k(x_k,u_k(x_k),w_k)+J_{k+1}(f_k(x_k,u_k,w_k)\right\}$$

然后给出动态规划算法流程图

Algorithm 1 Dynamic Programming Algorithm

Input: $J_N(x_N)=g_N(x_N)$,x_0
Output: optimal policy $\pi^*=\left\{\mu_0^*,\mu_1^*,\cdots,\mu_{N-1}^*\right\}$

1: **for** $k=N-1$ to 0 **do**
2: 计算最优策略 $u_k=\arg\min_{u_k\in U_k(x_k)} E\left\{g_k(x_k,u_k,w_k)+J_{k+1}\left(f_k(x_k,u_k,w_k)\right)\right\}$
3: $\mu_k^*(x_k)=u_k$
4: 计算最优成本 $J_k(x_k)=\min_{u_k\in U_k(x_k)} E\left\{g_k(x_k,u_k,w_k)+J_{k+1}\left(f_k(x_k,u_k,w_k)\right)\right\}$
5: **for** $k=0$ to $N-1$ **do**
6: 根据系统状态 x_k 计算最优控制 $u_k^*=\mu_k^*(x_k)$
7: 状态转移 $x_{k+1}=f_k(x_k,u_k,w_k)$
8: **return** $\pi^*=\left\{\mu_0^*,\mu_1^*,\cdots,\mu_{N-1}^*\right\}$

典型的动态规划逆序算法，先反向计算，然后正向求解。视频中说的策略是函数，或者叫状态到控制的映射，这个算法如果策略是函数的话是解释不通

的，只有策略是存在计算机内存中的状态到控制的映射才能解释得通，所以笔者认为这就是存在内存中的状态到控制的映射，先反向计算每一期每一个状态对应的最优控制，然后再把阶段向前推一期，利用刚刚计算得到的结果计算当前阶段的最优控制，就这么一直向前一直推到第一期，反向计算的时候计算倒数第二期的最优控制的时候用到了给定的终止状态对应的成本，就这样反向算完所有的最优控制，这些映射也就是最优策略。再根据给定的初始系统状态，依次向前计算相应期的最优控制，最后得到控制序列。

这种方法能够保证控制序列的全局最优性 (bellman 最优性原理)，但是需要使用非常多的计算机内存，大规模问题中的应用依然受到很大的限制。

实际上最优性的证明可以通过数学归纳法证明，证明过程如下：

Proof 给定了终止状态 x_N 的成本值 $J_N(x_N) = g_N(x_N)$，根据上文的定义有

$$J_{N-1}(x_{N-1}) = \min_{u_{N-1}\in U_{N-1}(x_{N-1})} E\left\{g_{N-1}(x_{N-1},u_{N-1},w_{N-1}) + g_N(f(x_{N-1},u_{N-1},w_{N-1}))\right\}$$
$$\min_{u_{N-1}\in U_{N-1}(x_{N-1})} E\left\{g_{N-1}(x_{N-1},u_{N-1},w_{N-1}) + J_N(f(x_{N-1},u_{N-1},w_{N-1}))\right\}$$

根据上式可以得到第 $N-1$ 期的最优控制

$$\mu^*_{N-1}(x_{N-1}) = \arg\min_{u_{N-1}\in U_{N-1}(x_{N-1})} E\left\{g_{N-1}(x_{N-1},u_{N-1},w_{N-1}) + J_N(f(x_{N-1},u_{N-1},w_{N-1}))\right\}$$

上述的最优成本 $J_{N-1}(x_{N-1})$ 和最优控制 $\mu^*_{N-1}(x_{N-1})$ 都是第 $N-1$ 期开始的子问题的最优解，所以不妨假设有第 k 期的最优解 $J_k(x_k), N-1 \geq k \geq 1$，系统状态 x_k 和控制策略 $\mu(\cdot)$ 都给定的时候，定义成本函数

$$Q_k(x_k|\mu(\cdot)) = \min_{u_k\in U_k(x_k)} E\left\{g_k(x_k,u_k,w_k) + \sum_{t=k+1}^{N-1} g_t(x_t,\mu(x_t),w_t)\right\}$$

当给定策略 $\mu(\cdot)$ 为最优控制策略 $\mu^*(\cdot)$ 时

$$\min_{u_k\in U_k(x_k)} Q_k(x_k|\mu^*(\cdot)) = J(x_k)$$

同时由于

$$Q_k(x_k|\mu(\cdot)) \geq Q_k(x_k|\mu^*(\cdot)), \forall\mu(\cdot)$$

存在

$$\min_{u_t\in U_t(x_t),t\geq k} Q_k(x_k|\mu(\cdot)) \geq \min_{u_k} Q_k(x_k|\mu^*(\cdot)), \forall\mu(\cdot)$$

考虑第 $k-1$ 期时有

$$\min_{u_t\in U_t(x_t),\forall t\geq k-1} E\left\{\sum_{t=k-1}^{N-1} g_t(x_t,u(x_t),w_t)\right\} \leq \min_{k-1} Q_{k-1}(x_{k-1}|\mu_t(\cdot),t\geq k) \leq \min_{u_t\in U_t(x_t),t\geq k-1} Q_{k-1}(x_{k-1}|\mu(\cdot))$$

显然

$$\min_{u_t\in U_t(x_t),\forall t\geq k-1} E\left\{\sum_{t=k-1}^{N-1} g_t(x_t,u(x_t),w_t)\right\} = \min_{u_t\in U_t(x_t),t\geq k-1} Q_{k-1}(x_{k-1}|\mu(\cdot))$$

所以

$$\min_{u_t\in U_t(x_t),\forall t\geq k-1} E\left\{\sum_{t=k-1}^{N-1} g_t(x_t,u(x_t),w_t)\right\} \leq \min_{k-1} Q_{k-1}(x_{k-1}|\mu_t(\cdot),t\geq k) = J(x_{k-1})$$

因此 $J(x_{k-1})$ 也是第 $k-1$ 期的全局最优解。

证明完毕

动态规划是一个计算机很容易理解的算法，但是实现起来非常困难，第一个困难是随着问题规模增大带来的计算量与存储量增大，而且是与状态和控制的空间成指数增长的。比如在三维空间中开车，系统状态有六个维度，假定每个维度离散化 100 个值，状态空间就有 100^6 个状态，假定每个状态可以执行的控制集合大小为 $|U|$，那么每一个阶段最小化时需要计算某个状态在所有控制下的期望成本，计算每一个控制下的期望成本时需要做一次最小化操作，然后在所有控制中选择一个期望值最小的控制执行，这只是一个阶段的计算，需要进行 $|U|$ 次期望计算和 $100^6|U|$ 空间内的最小化计算，再在 $|U|$ 维的向量中选择最小的控制，如果需要进行多阶段计算，计算量程指数增长，存储同样会出现问题，这就是一直在说的动态规划维数灾的问题。

2.9 PRACTICAL DIFFICULTIES OF DP

还有一个问题是建模困难，使用动态规划时需要知道状态转移的概率分布，也就是需要知道 $f(x_k,u_k,w_k)$，而 w_k 在应用场景中很难得到，也就限制了动态规划的应用。

另一个问题就是动态规划的实时性，动态规划的实际应用一般是实时性场景，在问题产生之前你不知道这个问题的参数和结构是什么样的，比如你需要让车去运货，但是派车需求产生之前你不知道什么地方需要货，或者问题的数据会随着控制被执行而改变，比如你派出了一辆车，这辆车执行任务时产生了更多的派车需求，这就需要立刻进行计算，但是动态规划的计算速度没有这么快，预先计算的 offline 方案无法使用，就需要一个 online 的算法，以上类似的场景动态规划就无法使用了。因此就需要考虑对动态规划算法做近似，或者考虑使用仿真的方法解决建模困难的问题。

2.10 A MAJOR IDEA: COST APPROXIMATION

近似成本不是唯一的近似思路，但是是一个很主要的近似思路，也就是使用一个更容易计算的问题的最优成本 $\tilde{J}_k$ 做近似函数来代替最优成本函数 J_k(简

化可以通过丢掉一些动态性和概率完成)，然后用这个替代函数 $\tilde{J}_k$ 求最优控制代替原问题的最优控制。

另一个例子是当状态空间特别大的时候，J_k 会非常多，这时候使用参数化的 $\tilde{J}_k$ 代替 J_k，在比较小的参数空间内搜索得到近似效果最好的替代函数 $\tilde{J}_k$，强化学习，神经动态规划和近似动态规划都是通过这种方式近似 cost to go 的。

第三种近似方法是一个很简单也很高效的方法，叫做 roll-out 方法，roll-out 方法使用比如启发式策略之类的次优策略成本 $\tilde{J}_k$ 来近似最优策略的成本，然后用这个近似的次优成本来计算近似的最优控制。

2.11 ROLLOUT ALGORITHMS

roll-out 算法需要设计一个启发式策略，也叫基准策略，在观测到系统状态时根据这个策略计算采取某动作后可能出现的系统状态对应的成本，再使用 bellman 方程选择要执行的控制。roll-out 算法设计的一个原则就是：启发式策略可以不是最好的，但是一定不可以是最坏的，而且至少不能比基准策略更坏，从这个启发式策略出发，通过前向仿真计算这个策略在控制执行后系统状态的成本，选择控制，一般情况下会得到一个有意义而且接近最优控制的策略，这种方法很适合在线计算，因为得到当前状态后，执行某控制可能得到的后续状态也可以预测到，就可以使用仿真得到相应的成本，然后选择控制，当然这种方法有一个潜在的问题，就是前向仿真计算策略成本的时候需要大量的计算。

如果你需要解决的是随机问题，你可以尝试使用蒙特卡罗模拟来计算成本，如果这是一个确定性的问题，就不需要使用蒙塔卡罗模拟而是直接计算成本，同时这种方法是一种解决离散优化问题的很重要的方法，如果你需要解决一个离散优化问题，你可以把决策变成多阶段的决策，然后使用启发式策略在一个状态轨迹仿真的过程中计算指定点的剩余路径成本获得剩余的控制方案，也就是剩下的解。

在控制理论中有一个非常重要的方法-模型预测控制 (Model Predictive Control, MPC)，控制系统的设计由于模型预测控制获得了很大的发展，模型预测控制就是一种在特殊的系统中特定情况下使用特定启发式策略的 roll-out 算法，它已经在控制理论中获得了很大的成功，所以大家都认为 roll-out 也是一种非常有效的技术。

2.12 INFINITE HORIZON PROBLEMS

以上讲的内容都是有限期问题，在求解有限期问题时，可以考虑对问题做近似然后求解，这只是一种比较基本的方法，也没有太多好的理论来解决有限期问题了，而无限期问题中就有很多比较有趣的理论。

无限期问题与有限期问题基本相同，只有两个地方不一样，第一个是无限期问题有无穷多个阶段，第二个时无限期系统是平稳的，系统平稳是指系统状态方程不依赖于时间 k，无论什么阶段系统演变都以相同的规律进行，如果满足了以上两个条件，系统就是无限期系统了。

求解无限期问题主要有两种思路，最小化总成本与最小化平均成本，主要有两种方法，使用折扣指标函数与阶段截断，其中最小化平均成本无论是截断法还是折扣法都比最小化总成本更难，事实上最小化平均成本的近似理论还在研究中，所以这个课程就不会涉及最小化平均成本的内容。

无限期问题与有限期问题相似，观测到系统状态 x_k 时执行一个控制 u_k，在随机扰动 w_k 的作用下付出即时成本 $g_k(x_k, u_k, w_k)$ 并观测到新的系统状态 x_{k+1}，然后继续进行控制的执行与新状态的观测。而这个控制是没有停止时间的，会一直进行下去，控制的目标就是成本最小，如上述累加总成本最小或者平均成本最小，此处给出累加总成本最小的指标函数 $J(x_k) = \lim\limits_{N\to\infty} E\left\{\sum\limits_{t=k}^{N-1} g_t(x_t, u_t, w_t)\right\}$，我们的目标就是最小化 $J(x_k)$，但由于即时成本 $g_k(x_k, u_k, w_k)$ 可能取到任何实数，所以 $J(x_k)$ 很可能是一个无界的函数 (几乎所有情况下都会发生)，这就没有办法分析与求解了，所以就需要采取措施应对这种情况，目前主要有两种方法处理，使用折扣系数与截断阶段，截断阶段是把趋于 ∞ 的 N 截断，令 N 等于一个固定的值，将无限期问题近似转化为有限期问题，然后求解。另一种方法，折扣因子是把每一期的即时成本乘以一个折扣因子 $\alpha \in (0, 1)$，距当前期越远的即时成本乘以的 α 越多，也就是说即时成本数列收敛，这样就能控制比较远的期对应的成本逐渐趋近于 0，令总成本有一个上界。这两种方法的作用都是消除长远影响，令距现在比较远的期的成本不对当前期的决策产生影响，从而控制指标函数的界，给出折扣问题的指标函数定义 $J(x_k) = \lim\limits_{N\to\infty} E\left\{\sum\limits_{t=k}^{N-1} \alpha^{t-k} g_t(x_t, u_t, w_t)\right\}$

教授在课程中还给了一个随机最短路径的例子，并且说这是一个无限期问题的例子，笔者个人不太认同这种分类，但是还是要介绍一下随即最短路径问题。在一个图中，从一个节点出发经过很多节点与弧之后到达一个确定的节点，需要决策的是经过的节点与经过他们的顺序，目标是最小化经过的所有弧的总长度，这个问题中折扣因子 $\alpha = 1$，同时给出到达哪一个节点的决策后不会确定地到达想要到达的节点，这就是随机最短路径的随机性所在，同时由于随机性与图的特殊性，可能会导致很多步之后才到终端节点，也就是从阶段数量不固定，极端情况下可能会出现无穷阶段的情况，这应该也是教授把这个问题归于无限期问题的原因。

上面讲了两种让指标函数有界的方法，还有一种方法是最小化平均成本，即总成本除以 N，指标函数变成 $\lim\limits_{N\to\infty} \frac{1}{N} E\left\{\sum\limits_{t=0}^{N-1} g_t(x_t, u_t, w_t)\right\}$，此时指标函数就变成有界函数，可以使用各种分析与求解方法处理了，但是由于时间有限，这个课程就不介绍了。

在我们的生活中很少 (甚至是完全) 不会产生无限期问题，那么为什么要研究无限期问题呢。原因是无限期问题在数学上更优雅，我们可以把注意力放在平稳策略上，这样就可以在任何阶段都使用相同的策略进行控制，大量的研究产生了大量有效的算法，很多近似理论也是为无限期问题服务的，所以我们需要研究无限期问题。

与非平稳策略可能在不同阶段使用不同的策略不同，平稳策略指的是在不同的阶段都使用同一个相同的策略。

无限期问题主要有如下四个种类，分类标准包括：总成本与平均成本、折扣与非折扣、指标函数与每期成本是否有界、阶段数量是否固定。

1. 最小化总成本
   - 折扣问题
   - 随机最短路径问题
   - 每期平均成本无界的折扣问题和非折扣问题
2. 平均成本问题

2.13 DISCOUNTED PROBLEMS/BOUNDED COST

上面介绍了无限期问题的几种类型，这个课程主要关心的是成本与指标函数有界的折扣无限期问题，这是一种比较有代表性的问题，

首先介绍平稳系统，如果一个系统每一个阶段的系统状态都在同一个状态空间内取值，随机干扰 w 在每一阶段都有相同的概率分布，那么这个系统就叫做平稳系统，与上文的平稳策略相似，系统/策略的特性与阶段无关，任何阶段都具有相同的特性。

现在有一个平稳系统，策略是一个函数序列，执行一个控制的时候会受到随机干扰 w 的影响，如果在相同状态下多次执行这个控制，再对相应的成本取平均值就可以得到这个状态下执行该控制的期望即时成本，把各阶段的期望即时成本乘以折扣因子累加起来就是整个策略的累加期望成本，也就是策略 π 的总成本 J_π，可以知道这个问题的最优成本就是在策略空间中最小化策略成本的最优解 $(J^*(x) = \min_\pi J_\pi(x))$。假设 g 有上界 M，那么 $|g| \le M$ 也成立，则指标函数 J 是一个有界函数，可以计算得到策略 π 的累加期望成本有上界 $\frac{M}{1-\alpha}$，即 $|J_\pi(x)| \le \frac{M}{1-\alpha}$，证明过程其实很简单，就是一个等比数列计算的过程，如下：

Proof 已知 $|g| \le M$，$J_\pi(x_k) = \lim_{N\to\infty} E\left\{\sum_{t=k}^{N-1} \alpha^{t-k} g_t(x_t, u_t, w_t)\right\}$，假设总成本不受到随机因素 w_k 的影响，有

$$J_\pi(x_k) = \lim_{N\to\infty} \sum_{t=k}^{N-1} \alpha^{t-k} g_t(x_t, u_t, w_t) \le \lim_{N\to\infty} \sum_{t=k}^{N-1} \alpha^{t-k} M$$

由于

$$\begin{aligned}\lim_{N\to\infty}\sum_{t=k}^{N-1}\alpha^{t-k}M =& M\lim_{N\to\infty}\sum_{t=k}^{N-1}\alpha^{t-k}\\ =& M\lim_{N\to\infty}\{1+\alpha+\alpha^2+\cdots+\alpha^{N-1-k}\}\\ =& M\lim_{L\to\infty}\{1+\alpha+\alpha^2+\cdots+\alpha^{L}\}\\ =& \frac{M}{1-\alpha}\end{aligned}$$

所以有 $J_\pi(x_k)\le\frac{M}{1-\alpha}$，$|J_\pi(x_k)|\le\frac{M}{1-\alpha}$ 也是成立的，因为 k 取任何值时都具备这个性质，所以也可以写成 $|J_\pi(x)|\le\frac{M}{1-\alpha}$

证明完毕

在这个问题中，状态 x 和控制 u 是连续的还是离散的都不重要，比较重要的是函数 J 与成本 g 是不是有界，实际解决问题的时候控制与状态会有连续与离散不同的情况，但是这不会产生太大的影响，会产生影响的就是函数是否有界。假设状态空间有限，则系统为有限空间马尔科夫链，当状态空间有限时，系统的状态在状态空间内随即移动，控制同样在一个有限的控制集合内取值，这类问题通常被叫做马尔科夫决策过程，或者叫 MDP，有些人研究的 MDP 在无限空间内，但是我后面提到 MDP 的时候，我说的是有限空间 MDP，近似的时候也需要离散化状态和控制保证 MDP 是一个有限空间 MDP，然后做近似，这也就是系统模型很重要的原因，同时人工智能领域的研究者也都假设或者构造有限空间的 MDP。

2.14 SHORTHAND NOTATION FOR DP MAPPINGS

上文给出的符号定义很复杂也很繁琐，对分析是非常不利的，现在我们要引入一种比较间接的符号来描述问题与动态规划算法，考虑一个无限期折扣平稳系统，最优成本函数定义为 $J(x_0)=\min\limits_{u_t,t\in[0,\infty]}\lim\limits_{N\to\infty}E\left\{\sum\limits_{t=0}^{N-1}\alpha^t g_t(x_t,u_t,w_t)\right\}$ 递推式有 $J(x_0)=\min\limits_{u_0}E\{g_0(x_0,u_0,w_0)+\alpha J(f(x_0,u_0,w_0))\}$，更一般地，有 $J(x_k)=\min\limits_{u_k}E\{g_k(x_k,u_k,w_k)+\alpha J(f(x_k,u_k,w_k))\}$，引入算子 T 并且重新定义成本函数 $(TJ)(x_k)=\min\limits_{u_k}E\{g_k(x_k,u_k,w_k)+\alpha J(f(x_k,u_k,w_k))\}$，算子指的是从函数到函数的映射，即从当前阶段的最优成本函数可以映射到前一阶段的最优成本函数，同理，无限期有界折扣问题成本函数 J 对于策略 μ 有算子 T_μ 的定义 $(T_\mu J)(x_k)=E\{g_k(x_k,\mu(x_k),w_k)+\alpha J(f(x_k,\mu(x_k),w_k))\}$，无论是策略成本还是总成本，当前期成本函数有上界，算子 T 一定可以产生一个有上界的前一期的成本函数。

这种结构对于最优成本算子 T 和策略算子 T_μ 都是成立的，同时所有的折

扣问题的理论都可以使用这种简洁的符号表示，这种表达方法可以让分析与证明时主要关注问题与算法的性质而不是其他更细节的问题。

2.15 FINITE-HORIZON COST EXPRESSIONS

对于有限期折扣问题，有一个 N 阶段策略 $\pi_0^N = \{\mu_0, \mu_1, \cdots, \mu_{N-1}\}$，更一般地，$\pi_k^N = \{\mu_k, \mu_{k+1}, \cdots, \mu_{N-1}\}$，在策略 π_0^N 下有成本函数

$$\begin{aligned} J_{\pi_0^N}(x_0) =& E\left\{ g_0(x_0, \mu(x_0), w_0) + \sum_{l=1}^{N-1} \alpha^l J(f(x_l, \mu(x_l), w_l)) \right\} \\ =& E\left\{ g_0(x_0, \mu(x_0), w_0) + \alpha J_{\pi_1^N}(x_1) \right\} \\ =& (T_{\mu_0} J_{\pi_1^N})(x_0) \\ =& (T_{\mu_0} T_{\mu_1} \cdots T_{\mu_{N-1}} J)(x_0) \end{aligned}$$

同理可以写出 $J_{\pi_k^N}(x_k) = (T_{\mu_k} T_{\mu_{k+1}} \cdots T_{\mu_{N-1}} J)(x_k)$

上述的成本函数表达式对于任意策略 $\pi_0^N = \{\mu_0, \mu_1, \cdots, \mu_{N-1}\}$ 都成立，如果使用平稳策略 $\pi_0^N = \{\mu, \mu, \cdots, \mu\}$，成本函数可以写成 $J_{\pi_0^N} = T_\mu^N J$，更一般地，有 $J_{\pi_k^N} = T_\mu^{N-k} J$，其中 T_μ^{N-k} 表示将 N 个算子 T_μ 作用在 J 上或将算子 T_μ 在 J 上作用 N 次。

符号 T 与符号 T_μ 之间的联系是：$T = \min_\mu T_\mu$，所以根据上述理论，表达式 $T_N J = T(T^{N-1} J)$ 与上文的 bellman 最优方程是等价的。

2.16 "SHORTHAND" THEORY –A SUMMARY

给定一个无限期问题，假设终端成本为 0，即 $J_0(x) \equiv 0$，就有策略 π 的成本函数 $J_\pi(x) = \lim_{N\to\infty} (T_{\mu_0} T_{\mu_1} \cdots T_{\mu_N} J_0)$，如果考虑特殊情况，$\pi$ 是由 μ 构成的平稳策略，此时策略 μ 的成本函数可以写为 $J_\mu(x) = \lim_{N\to\infty} (T_\mu^N J_0)(x)$，bellman 方程的两种形式分别为：最优成本 bellman 方程 $J^* = TJ^*$ 和策略 μ 下的 bellman 方程 $J_\mu = T_\mu J_\mu$ 上文中给出过 T 和 T_μ 的关系：$T = \min_\mu T_\mu$，由这个关系可以得到策略 μ 的最优性条件：$T_\mu J^* = TJ^*$，实际上这个最优性条件笔者认为很难满足或者是判断是不是满足，因为最优算子 T，最优成本函数 J^* 都是未知的东西，也就没有判断最优性条件满足的依据。

动态规划的两个基本算法，值迭代和策略迭代，分别给出算法迭代公式。

值迭代：对于任意有界成本函数 J，有最优成本函数

$$J^*(x) = \lim_{k\to\infty}(T^k J)(x), \forall x$$

即从任意成本函数出发不断使用最优算子 T 最后都会收敛到最优成本函数。

策略迭代：对于一个给定的策略 μ^k，通过求解

$$J_{\mu^k} = T_{\mu^k} J_{\mu^k}$$

得到该策略的成本函数，然后使用

$$T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k}$$

改进策略，改进后继续评估-改进的循环，这就是策略迭代的策略评价和策略改进过程，其中策略评价其实是寻找给定策略 μ_k 的不动点的操作，策略改进是由于策略评估后对于策略成本来说当前策略 J_{μ^k} 不是最优策略了，所以需要计算当前策略成本对应的最优策略 T，然后将其迭代修改为当前策略 $T_{\mu^{k+1}}$，其实就是做一个 $T_{\mu^{k+1}} = \arg\min_{T_\mu} T_\mu J_{\mu^k}$ 的操作。

2.17 TWO KEY PROPERTIES

T 和 T_μ 有两个很重要的性质，第一个是单调性，对于任何成本函数 J 和 J'，如果有 $J \le J'$，那么在使用相同的算子 T 进行映射的时候这个不等式关系依然能够被保存下来，即 $(TJ)(x) \le (TJ')(x)$ 和 $(T_\mu J)(x) \le (T_\mu J')(x)$，此时说算子具有单调性，这是一个动态规划中很普遍的性质，不仅是折扣问题，对于所有的动态规划问题都成立。

第二个性质比较特殊，只针对于折扣问题，叫做恒定位移性 (Constant Shift property)，对于任意的成本函数 J，任意的实数标量 r 和任意的策略 μ，都有 $(T(J+re))(x) = (TJ)(x) + \alpha r, \forall x$ 和 $(T_\mu(J+re))(x) = (T_\mu J)(x) + \alpha r, \forall x$，需要特殊提到的是，$e$ 是一个单位函数，即 $e(x) \equiv 1$

上述两个性质，单调性所有动态规划模型都具备，恒定位移性质只有折扣模型具备，除了这两个性质外，折扣模型还有另一个比较重要的性质：收缩性，这个性质后面会提到。

2.18 CONVERGENCE OF VALUE ITERATION

这页要证明动态规划值迭代算法的收敛性，课件上的证明有点简略，我想在收敛性的证明上写的详细一点，先根据笔者的理解给出动态规划收敛性的证明，然后再分析课件的过程。

动态规划的基本算法有两个，值迭代和策略迭代，策略迭代还包括策略评

价和策略改进，也就是某策略下的值迭代和策略迭代，这三种迭代算法的收敛性主要包括三个部分：是否收敛，收敛的解是否唯一，收敛速度是什么样的。这三种迭代算法，其实就是求 bellman 方程不动点的迭代算法，所以这里要用泛函的角度分析他们的收敛性和不动点的唯一性。

首先给出几个定义：

Definition ***: metric space***

一个 ***metric space*** 由一个二元组 $\langle M, d\rangle$ 组成，其中 M 表示集合，d 表示 M 的一个 *metric*，即映射 $d : M \times M \to \mathbb{R}$ 。其中 d 满足如下的距离的三条公理：对于任意 $x, y, z \in M$ ，有

1. *(非负性)* $d(x, y) \geq 0$ ，且 $d(x, y) = 0 \Leftrightarrow x = y$
2. *(对称性)* $d(x, y) = d(y, x)$
3. *(三角不等式)* $d(x, z) \leq d(x, y) + d(y, z)$

Definition ***: Cauchy sequence***

设 x_n 是距离空间 X 中的点列，如果对于任意的 $\epsilon > 0$，存在自然数 N，当 $m, n > N$ 时，$|x_n - x_m| < \epsilon$，称 x_n 是一个 *Cauchy* 列。

Definition ***: complete metric space***

一个 *metric space* $\langle M, d\rangle$ 是完备的 *(*或者说是 *Cauchy* 的*)*，当且仅当所有在 M 中的 *Cauchy* 序列，都会收敛到 M 中。即在完备的 *metric space* 中，对于任意在 M 中的点序列 $a_1, a_2, a_3, \ldots \in M$ ，如果序列是 *Cauchy* 的，那么该序列收敛于 M*,* 即 $\lim_{n\to\infty} a_n \in M$

Definition ***: contraction mapping***

In mathematics, a contraction mapping, or contraction or contractor, on a metric space $\langle M, d\rangle$ *is a function f from M to itself, with the property that there is some nonnegative real number* $0 \leq k < 1$ *such that for all x and y in M,*

$$d(f(x), f(y)) \leq kd(x, y)$$

The smallest such value of k is called the Lipschitz constant of f. Contractive maps are sometimes called Lipschitzian maps. If the above condition is instead satisfied for $k \leq 1$*, then the mapping is said to be a non-expansive map.*

简单介绍一下这几个定义，Metric Space 表示的是某个集合中任意两个点之间的距离都被定义了，那么这些点的集合就叫做 Metrix Space，这些距离被叫做这个 space 的一个 metric，同一个 space 可以有无数个 metric。

直观地讲，Cauchy sequence 指的是如果一个 metric space 中的序列上的点随着点的序号变大，两个点之间的距离 (定义 metrix space 时定义的距离) 越来越近直到无限接近的趋势。

complete metric space 是说某个 metric space 中所有 Cauchy sequence 最后都会收敛于该 metric space 中的元素，这个 metric space 才是 complete metric space，所以想要证明一个 metric space 不是 complete matric space 很简单，只要找到一个反例就可以了。

contraction mapping 是说某个 matric space 中的映射 f 如果满足任意两

个点经过映射后产生的两个新的点距离要比映射前的点距离要近，这个映射 f 就是该 matric space 中的一个 contraction mapping，满足条件的最小的 k 被叫做 f 的 Lipschitz constant，k 越小说明压缩得越厉害。

给出压缩映射定理：

Theorem *: **Contracting mapping theorem***

对于一个 *complete metric space* $\langle M, d\rangle$，如果 $f : M \mapsto M$ 是它的一个压缩映射，那么

1. 在该 *complete metric space* 中，存在唯一的点 x^* $f(x^*) = x^*$

2. 并且，对于任意的 $x \in M$，定义序列 $f^2(x) = f(f(x))$，$f^3(x) = f(f^2(x)), \cdots, f^n(x) = f(f^{n-1}(x))$, 该序列会收敛于 x^* 即 $\lim_{n\to\infty} f^n(x) = x^*$

定理的证明分成两部分，先证明定理的第二条，即序列收敛，然后证明序列收敛于唯一的点，证明如下：

Proof 序列收敛：

对于任意 $x, y \in M$，根据三角不等式

$$\begin{aligned} d(x,y) &\le d(x,f(x)) + d(f(x),y) \\ &\le d(x,f(x)) + d(f(x),f(y)) + d(f(y),y) \end{aligned}$$

根据压缩映射的定义

$$d(x,f(x)) + d(f(x),f(y)) + d(f(y),y) \le d(x,f(x)) + kd(x,y) + d(f(y),y)$$

根据对称性

$$d(x,f(x)) + kd(x,y) + d(f(y),y) = d(f(x),x) + kd(x,y) + d(f(y),y)$$

所以，综上：

$$\begin{aligned} d(x,y) &\le d(f(x),x) + kd(x,y) + d(f(y),y) \\ \Rightarrow d(x,y) &\le \frac{d(f(x),x) + d(f(y),y)}{1-k} \end{aligned}$$

序列 $f^n(x)$ 中的任意两个点 $f^M(x), f^N(x)$，满足

$$\begin{aligned} d(f^M(x), f^N(x)) &\le \frac{d(f^{M+1}(x), f^M(x)) + d(f^{N+1}(x), f^N(x))}{1-k} \\ &\le \frac{k^M d(f(x),x) + k^N d(f(x),x)}{1-k} \\ &= \frac{k^M + k^N}{1-k} d(f(x),x) = (k^M + k^N)\frac{d(f(x),x)}{1-k} \end{aligned}$$

对于给定的初始点 x 和 $k \in [0,1)$，上式左项是一个常数，右项在 $M, N \to \infty$ 时是一个无穷小量，所以有

$$\frac{d(f(x),x)}{1-k} \lim_{M,N\to\infty}(k^M + k^N) = 0$$

即 $\lim_{M,N\to\infty} d(f^M(x), f^N(x)) = 0$，所以序列 $f^n(x)$ 收敛。

证明完毕

Proof 不动点唯一性：

由于序列 $f^n(x)$ 是收敛的同时 $f^n(x)$ 处于 *complete metric space* 中，假设它收敛于 *metric space* 中某个不动点 $x^* = \lim_{n\to\infty} f^n(x)$，根据压缩映射的定义可知 f 连续，所以 $f(x^*) = f(\lim_{n\to\infty} f^n(x)) = \lim_{n\to\infty} f(f^n(x)) = \lim_{n\to\infty} f^{n+1}(x) = x^*$。

证明不动点存在后用反证法证明 x^* 的唯一性，假设有两个点 x^*, y^* 满足 $f(x^*) = x^*, f(y^*) = y^*$，那么 $0 < d(x^*, y^*) = d(f(x^*), f(y^*)) \le kd(x^*, y^*) < d(x^*, y^*)$ 得到 $d(x^*, y^*) < d(x^*, y^*)$，与假设矛盾，因此假设不成立，$x^*$ 是唯一的不动点。

证明完毕

由于 $x^* = f(x^*)$，所以 $d(f^n(x), x^*) = d(f^n(x), f(x^*))$，根据压缩映射的定义，可以得到 $d(f^n(x), f(x^*)) \le kd(f^{n-1}(x), x^*) \le k^n d(x, x^*)$，所以序列 $f^n(x)$ 以线性速率 k 收敛。

现在分析策略评价、值迭代和策略迭代的收敛性：假设状态集合 $S = \{s_1, s_2, \cdots, s_n\}$，行动集合 $A = \{a_1, a_2, \cdots, a_n\}$，定义状态值向量函数 $\boldsymbol{v} = [v(s_1), v(s_2), \cdots, v(s_n)]^T$ 在值函数空间 $\mathcal{V}$ 中，$\mathcal{A}$ 的距离函数 $d(\boldsymbol{v_1}, \boldsymbol{v_2}) = \max_{s\in\mathcal{S}} |\boldsymbol{v_1}(s) - \boldsymbol{v_2}(s)|$，由于 $\mathcal{V} = \mathcal{R}^n$ 所以所有序列的点一定都在空间 $\mathcal{V}$ 中，因此 $\langle \mathcal{V}, d\rangle$ 是一个 complete metric space。定义策略函数向量 $\boldsymbol{\pi} = [\mu(s_1), \mu(s_2), \cdots, \mu(s_n)]^T$ 属于行动空间 $\mathcal{A}$, 定义 $\mathcal{A}$ 中的距离函数 $d(\mu_1, \mu_2) = \max_{s\in\mathcal{S}} |(||a_1(s) - a_2(s)||_2)|$

策略评价：

bellman 期望方程向量形式为 $\mathbf{u}_{new} = T^\pi(\mathbf{u}) = \mathcal{R}^\pi + \gamma\mathcal{P}^\pi\mathbf{u}$，其中 $\mathcal{P}^\pi$ 是策略 π 下状态转移的概率分布函数，可以计算得到 $\mathbf{u}$ 与 $\mathbf{v}$ 使用算子 T^π 后新的点距离为

$$\begin{aligned} d(T^\pi(\mathbf{u}), T^\pi(\mathbf{v})) &= \|(\mathcal{R}^\pi + \gamma\mathcal{P}^\pi\mathbf{u}) - (\mathcal{R}^\pi + \gamma\mathcal{P}^\pi\mathbf{v})\|_\infty \\ &= \|\gamma\mathcal{P}^\pi(\mathbf{u} - \mathbf{v})\|_\infty \\ &\le \|\gamma\mathcal{P}^\pi\|\mathbf{u} - \mathbf{v}\|_\infty\|_\infty \\ &\le \gamma\|\mathbf{u} - \mathbf{v}\|_\infty \\ &= \gamma d(\mathbf{u}, \mathbf{v}) \end{aligned}$$

所以 T^π 是一个压缩映射，由压缩映射定理 (Contracting mapping theorem) 可以得到

1. 值向量空间 $\mathcal{V}$ 中存在唯一的点 $\mathbf{v}_\pi^*$ 满足 $\mathbf{v}_\pi^* = T^\pi(\mathbf{v}_\pi^*)$
2. 对于任意的 $\mathbf{v} \in \mathcal{V}$ 在映射 T^π 的作用下都会收敛于 $\mathbf{v}_\pi^*$，即 $\lim_{n\to\infty} T_\pi^n(\mathbf{v}) = \mathbf{v}_\pi^*$
3. 策略评价迭代以线性速率 γ 收敛于 $\mathbf{v}_\pi^*$

值迭代：
bellman 最优方程 (值迭代) 的矩阵形式迭代式：

$$\mathbf{u}_{new} = T^*(\mathbf{u}) = \max_{a\in\mathcal{A}} \mathcal{R}^a + \gamma\mathcal{P}^a\mathbf{u}$$

与策略评价迭代式相似，可以得到

$$d(T^*(\mathbf{u}), T^*(\mathbf{v})) \le \gamma d(\mathbf{u}, \mathbf{v})$$

所以 T^* 也是压缩映射，由压缩映射定理可以得到相同的结论：

1. 值向量空间 $\mathcal{V}$ 中存在唯一的点 $\mathbf{v}^*$ 满足 $\mathbf{v}^* = T^*(\mathbf{v}^*)$
2. 对于任意的 $\mathbf{v} \in \mathcal{V}$ 在映射 T^* 的作用下都会收敛于 $\mathbf{v}^*$，即 $\lim_{n\to\infty} (T^*)^n(\mathbf{v}) = \mathbf{v}^*$
3. 值迭代以线性速率 γ 收敛于 $\mathbf{v}^*$

策略改进：
策略改进矩阵形式表达式：$\boldsymbol{\pi_{new}} = I^*(\boldsymbol{\pi}) = \arg\max_\pi \mathcal{R}^\pi + \gamma\mathcal{P}^\pi\mathbf{v}$
与策略评价迭代式相似，可以得到

$$d(I^*(\pi_{\mathbf{1}}), I^*(\pi_{\mathbf{2}})) \le \gamma d(\pi_{\mathbf{1}}, \pi_{\mathbf{2}})$$

所以 I^* 也是压缩映射，由压缩映射定理可以得到相同的结论：

1. 策略空间 Π 中存在唯一的点 π^* 满足 $\pi^* = I^*(\pi^*)$
2. 对于任意的 $\pi \in \Pi$ 在映射 I^* 的作用下都会收敛于 π^*，即 $\lim_{n\to\infty} (I^*)^n(\pi) = \pi^*$
3. 值迭代以线性速率 γ 收敛于 π^*

课件中想要证明经过无数次算子 T 的作用后成本函数一定会收敛于最优成本函数，但是笔者认为这一页的假设基础是有问题的，问题在于不能用给定策略 π 对应的算子 T_π 进行迭代，而应该使用 bellman 最优方程的最优算子 T 进行迭代，下面笔者按照教授的思路使用最优算子 T 进行迭代证明值迭代的收敛性。

考虑一个成本函数 $J \equiv 0$ 有界的无限期折扣问题，想要证明

$$J^*(x) = \lim_{k\to\infty} (T^k J)(x), \forall x$$

Proof 值迭代算法收敛性：

首先给出无限期折扣问题的 *bellman* 最优性方程

$$J(x_0)^* = \min_{u_l \in U(x_l), l\in 0,1,\cdots} \lim_{k\to\infty} E\left\{\sum_{k=0}^{\infty} \alpha^k g(x_k, u_k, w_k)\right\}$$
$$= \min_{u_l \in U(x_l), l\in 0,1,\cdots} \lim_{k\to\infty} E\left\{\sum_{k=0}^{N} \alpha^k g(x_k, u_k, w_k) + \sum_{k=N}^{\infty} \alpha^k g(x_k, u_k, w_k)\right\}$$

已知 $|g| \le M$，所以上式满足

$$J(x_0)^* = \lim_{N\to\infty} \min_{\substack{u_l \in U(x_l),\\ l\in 0,1,\cdots}} E\left\{\sum_{k=0}^{N} \alpha^k g(x_k, u_k, w_k) + \sum_{k=N}^{\infty} \alpha^k g(x_k, u_k, w_k)\right\}$$
$$\le \lim_{N\to\infty} \min_{\substack{u_l \in U(x_l),\\ l\in 0,1,\cdots,N-1}} E\left\{\sum_{k=0}^{N} \alpha^k g(x_k, u_k, w_k) + \frac{\alpha^k M}{1-\alpha}\right\}$$
$$= \lim_{N\to\infty} \min_{\substack{u_l \in U(x_l),\\ l\in 0,1,\cdots,N-1}} \left\{E\left\{\sum_{k=0}^{N} \alpha^k g(x_k, u_k, w_k)\right\} + \frac{\alpha^k M}{1-\alpha}\right\}$$
$$= \lim_{N\to\infty} \min_{\substack{u_l \in U(x_l),\\ l\in 0,1,\cdots,N-1}} E\left\{\sum_{k=0}^{N} \alpha^k g(x_k, u_k, w_k)\right\} + \lim_{k\to\infty} \frac{\alpha^k M}{1-\alpha}$$
$$= \lim_{N\to\infty} \min_{\substack{u_l \in U(x_l),\\ l\in 0,1,\cdots,N-1}} E\left\{\sum_{k=0}^{N} \alpha^k g(x_k, u_k, w_k)\right\} = \lim_{N\to\infty} (T^N J)(x_0)$$

证明完毕

其实这个证明笔者个人理解应该是一个定义，上述过程只是对这个定义的推导，有待商榷，如果是不从泛函的角度证明收敛，可以证明算子 T 作用无数次后继续使用算子 T 不会对成本函数造成影响，上面的过程证明了无数次算子会导致值向量收敛于某值，下面证明这个值是唯一的。

Proof 不动点唯一：

已知 $J_N^*(x_0) \lim_{N\to\infty} (T^N J)(x_0)$，假设有不唯一的不动点，存在 $N \le M$ 且与相应的两个不同的不动点
$J_N^*(x_0) = \lim_{N\to\infty} (T^N J)(x_0)$ 和 $J_M^*(x_0) = \lim_{N\to\infty} (T^M J)(x_0)$，由定义可知：

$$J_N^*(x_0) - J_M^*(x_0) = \lim_{\substack{N\to\infty\\ M\to\infty}} \min_{\substack{u_l \in U(x_l),\\ l\in N, N+1,\cdots,M-1}} E\left\{\sum_{k=N}^{M-1} \alpha^k g(x_k, u_k, w_k)\right\} = 0$$

这与 $J_N^*(x_0)$ 和 $J_M^*(x_0)$ 是两个不同的点矛盾，假设不成立，所以不动点唯一。

证明完毕

2.19 BELLMAN' S EQUATION

这一页想证明 bellman 方程的解 J^* 满足方程 $J^* = TJ^*$，即

$$J^*(x) = \min_{u \in U(x)} E\{g(x,u,w) + \alpha J^*(f(x,u,w))\}$$

证明过程如下：

Proof 由上一页内容可知，$J^*(x) = (T^k J_0) + \frac{\alpha^k M}{1-\alpha}$，可以得到

$$J^*(x) - \frac{\alpha^k M}{1-\alpha} = (T^k J_0)$$

同时由于 $\frac{\alpha^k M}{1-\alpha} \geq 0$，所以可知

$$(T^k J_0) \leq J^*(x) \leq J^*(x) + \frac{\alpha^k M}{1-\alpha}$$

综上，有不等式关系

$$J^*(x) - \frac{\alpha^k M}{1-\alpha} \leq (T^k J_0) \leq J^*(x) + \frac{\alpha^k M}{1-\alpha}$$

已知 $J_0(x) \equiv 0$ 和 $M \geq |g(x,u,w)|$，对上面不等式使用算子 T，根据 *Monotonicity* 和 *Constant Shift*，可以得到结论：

$$T(J^*(x) - \frac{\alpha^k M}{1-\alpha}) \leq T((T^k J_0)) \leq T(J^*(x) + \frac{\alpha^k M}{1-\alpha})$$
$$\Rightarrow (TJ^*)(x) - \frac{\alpha^{k+1} M}{1-\alpha} \leq (T^{k+1} J_0)(x) \leq (TJ^*)(x) + \frac{\alpha^{k+1} M}{1-\alpha}$$

当 $k \to \infty$ 时，$\frac{\alpha^{k+1} M}{1-\alpha} = 0$，所以有

$$\lim_{k\to\infty} (TJ^*)(x) \leq \lim_{k\to\infty} (T^{k+1} J_0)(x) \leq \lim_{k\to\infty} (TJ^*)(x)$$
$$\Rightarrow \lim_{k\to\infty} (TJ^*)(x) = \lim_{k\to\infty} (T^{k+1} J_0)(x) = J^*(x)$$

所以有 $J^* = TJ^*$

证明完毕

2.20 THE CONTRACTION PROPERTY

本页想要证明 bellman 最优方程的算子 T 和 bellman 策略评价方程的算子 T_μ 都是压缩映射，在前面已经证明过了，这里就不重复证明了，顺便说一下，只有证明了这两个算子是压缩映射的情况下，压缩映射定理才有可能生效，即值迭代与策略评价能够收敛于唯一的不动点。

2.21 NEC. AND SUFFICIENT OPT. CONDITION

本页证明的是策略迭代的最优性，即只有满足 $TJ^* = T_\mu J^*$ 时，策略 μ 才是最优策略，证明思路是从 $TJ^* = T_\mu J^*$ 出发，证明策略 μ 是最优策略，证明过程如下：

Proof 策略最优性充分性： 如果存在 $TJ^* = T_\mu J^*$，已知 $J^* = TJ^*$，可以得到

$$J^* = T_\mu J^*$$

显然，J^* 是算子 T_μ 的不动点，即 $J^* = J_\mu$，所以这时候策略 μ 是最优策略。

证明完毕

Proof 策略最优性必要性： 如果已知策略 μ 为最优策略，即 $J_\mu = J^*$，同时存在 $J_\mu = T_\mu J_\mu$，$J^* = TJ^*$, 可以得到

$$T_\mu J^* = J^* = TJ^*$$

所以 $T_\mu J^* = J^* = TJ^*$ 成立。

证明完毕

3

LECTURE 2: EXACT DYNAMIC PROGRAMMING ALGORITHMS

CONTENTS

3.1 LECTURE OUTLINE

上一次课程讲了 n 阶段有限期问题、无限期折扣问题与非折扣问题的精确动态规划求解，还介绍了他们的速记符，这次课要讲无限期折扣问题，包括它

的速记符表达形式、具体形式的折扣动态规划值迭代和不同形式的策略迭代与策略迭代过程中使用的短视策略。然后要简单介绍一下 Q-learning 算法，后面的课程会详细讲。然后要讨论更多与折扣问题不一样的问题的模型，当然这个课程我们的主要注意力会放在折扣问题上，因为折扣问题很简单，还可以得到一个比较好的结果，通过对折扣问题的研究可以得到一个比较好的 benchmark，接下来讲决策空间、状态空间与决策时间都连续的问题。然后我们不在讨论具体的动态规划了，要从更抽象的角度来讨论它，首先从抽象的角度对动态规划做一个综述，这样可以为动态规划开发一个统一的理论，而不是只将他们应用在折扣问题中，同时还要解释为什么一些结论只能应用在一部分模型上，另一些模型会导致这些结果失效。最后一个话题是异步算法，上面介绍的算法都是同时更新所有值的，但是如果有一个并行机构可以并行地执行值迭代、策略迭代和基于仿真的算法，那么就可以不同时地处理所有状态，一部进行算法迭代，最后，异步算法的值迭代和策略迭代的性能都是可以得到理论保证的。

这次课程的主要目录如下：

1. Review of discounted problem theory
2. Review of shorthand notation
3. Algorithms for discounted DP
4. Value iteration
5. Various forms of policy iteration
6. Optimistic policy iteration
7. Q-factors and Q-learning
8. Other DP models - Continuous space and time
9. A more abstract view of DP
10. Asynchronous algorithms

3.2 DISCOUNTED PROBLEMS/BOUNDED COST

考虑一个无限期问题，有一个平稳系统，可以根据状态 x_k 和控制 u_k 下在随机扰动 w_k 的作用下产生新的系统状态 x_{k+1}，这个状态转移会一直持续下去，状态方程为

$$x_{k+1} = f(x_k, u_k, w_k), k = 0, 1, \cdots$$

在控制这个系统时会有一个原则，这里被叫做策略，策略 π 是一个函数序列，每个阶段都有一个相应的函数，这些函数是状态 x 到控制 u 的映射，观测到系统状态 x 的时候策略 π 会给出一个控制 $\pi(x)$，我们要做的就是找到最优的映射关系 π^*。

如果我要使用策略 π 控制一个即时成本为 $g(x, u, w)$ 的无限期系统，那么

策略 π 下的折扣成本等于 $J_\pi(x_0)\lim\limits_{N\to\infty}\sum\limits_{k=0}^{N-1}\alpha^k g(x_k,\mu(k),w_k)$，由于这个总成本在随机扰动 w_k 下表现为随机总成本，因此想要对策略 π 进行优化需要最小化期望总成本 $J_\pi(x_0)\lim\limits_{N\to\infty}E\left\{\sum\limits_{k=0}^{N-1}\alpha^k g(x_k,\mu(k),w_k)\right\}$

这个表达式中有一个很重要的假设，就是折扣系数 α 的取值需要满足 $\alpha\in(0,1)$，很多情况下即时成本 $g(x,u,w)$ 的值是有上界 M 的，即 $|g(x,u,w)|\leq M$，对于有限期问题，直接累加成本函数最终得到的期望总成本也会有相应的上界 NM，但是对于无限期问题，当 $N\to\infty$ 时期望总成本的上界 $NM\to\infty$，这个时候期望总成本是没有办法作为最小化目标的，就需要采取一些措施让期望总成本有一个确定的实上界，大部分无限期问题中使用折扣系数来约束期望总成本的上界，即上面表达式的计算方式，由于有 $|g(x,u,w)|\leq M$，给定折扣系数 $\alpha\in(0,1)$ 后各期的成本序列就变成了一个等比序列，很容易得到这个序列收敛于 0 并且累加有界的结论，这时候就可以把折扣期望总成本作为优化目标进行求解了。

在这里再一次定义算子 $T,(TJ)(x)=\min\limits_{u\in U(x)}E\left\{g(x,u,w)+\alpha J(f(x,u,w))\right\},\forall x$，使用算子 T 与相关符号可以比较简单地表示动态规划的相关理论，上面的算子 T 是最优 bellman 方程的算子，相似地，可以给出策略 π 的算子 T_π 的定义：$(T_\pi J)(x)=\min\limits_{u\in U(x)}E\left\{g(x,\mu(x),w)+\alpha J(f(x,\mu(x),w))\right\},\forall x$，最优算子与策略算子的区别是，不断使用最优算子可以让成本函数最终收敛于最优成本 J^*，而不断使用策略算子最终会得到该策略下的最优成本 J_π^*

3.3 “SHORTHAND” THEORY –A SUMMARY

上文给出了两种算子的定义，对于任意的初始成本 J，不断使用算子最终都会让成本收敛于唯一解，即 J^* 和 J_π，从泛函的角度上来说，J^* 和 J_π 就是算子 T 和 T_π 的不动点，也就是 $J^*=TJ^*$ 和 $J_\pi=T_\pi J_\pi$ 的不动点，具体地说，每一个状态 x 都对应一个成本 $J(x)$，即 J 是一个 n 维的向量，n 为状态集合中状态的数量，每一个状态成本都满足 bellman 方程，这样就有了一个 $n\times n$ 的方程组，方程组的解即为最优成本或策略成本，而对任意初始成本不断使用算子迭代的过程只是求解这个方程组的迭代算法之一。

在实际问题中，n 的数量会非常巨大，这导致了方程组及其难以求解，假设你已经求得了该方程组的最优成本的解，就可以通过下面的最小化操作来获得最优策略 $\mu(x)\in\arg\min\limits_{u\in U(x)}E\left\{g(x,u,w)+\alpha J^*(f(x,u,w))\right\}$，这个时候获得的就是这个问题的最优策略，也就是说此时策略 μ^* 满足 $T_\mu J^*=TJ^*$ 与 $T_{\mu^*}=T$，而反过来，如果有一个策略 μ 满足这两个关系，那么这个策略就是最优策略，相关的证明在 Lecture 1 中已经给出了，这里就不重复了，同样的，值迭代与策略迭代的收敛性与最优性也已经证明过了，不在重复证明，这里只介绍他们的性质。

对于值迭代算法，任意有界的初始 J 有 $J^*(x) = \lim_{k\to\infty}(T^k J(x)), \forall x$，也就是说对任意有界初始 J 不断使用最优算子 T 一定能够收敛于最优成本。

对于策略迭代算法，对于任意的策略 μ^k，先计算该策略的成本，也就是算子 T_μ 的不动点 J_μ，这个过程被称作策略评价，然后使用 J_μ 计算 $T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k}$，也就是进行策略改善得到更好的策略 μ^{k+1}，重复进行策略评价和策略改进，最终会收敛于最优策略 μ^*。

值迭代与策略迭代的主要区别是：值迭代致力于寻找最优成本函数，然后使用贪婪策略给出最优策略，而策略迭代致力于寻找最优策略，直接从一个初始策略开始进行迭代直接找到最优策略。这两个方法是动态规划中非常重要也非常基础的算法，大量的近似动态规划算法都是在这两个算法的基础上发展而来的，下面要介绍一下这两种理论生效的重要性质，这些性质是所有的证明、理论与分析的基础。

3.4 MAJOR PROPERTIES

开始讲动态规划的算法之前，再重复一次动态规划很重要的性质：单调性和收缩性，下面详细说明：

1. 单调性：

   对于状态空间 X 中的任意满足 $J(x) \le J'(x)$ 关系的函数 J 和 J'，单调性算子 T 作用之后依然满足大小关系：

   $$(TJ)(x) \le (TJ')(x), \quad (T_\mu J)(x) \le (T_\mu J')(x), \quad \forall x \in X$$

   所有的动态规划算子都具有单调性，这个性质的证明可能涉及泛函的单调算子理论，由于水平有限，笔记中目前不做证明，等到看过相关内容后再回来补充证明过程。

2. 收缩性：

   对于状态空间 X 中任意有界函数 J，J' 和策略 μ，压缩算子 T 满足：

   $$\max_x |(TJ)(x) - (TJ')(x)| \le \max_x \alpha |J(x) - J'(x)$$
   $$\max_x |(T_\mu J)(x) - (T_\mu J')(x)| \le \max_x \alpha |J_\mu(x) - J'_\mu(x)$$

   有一个比较重要的事情需要注意，收缩性只有折扣问题算子才具备，证明在 lecture 1 里面已经给了，其他类型的问题没法证明具有收缩性。收缩性的直观解释就是两个序列的点在同一个动态规划算子的作用下距离越来越近，至于这个性质有什么用，后面用到的时候再说。

关于收缩性，做一点补充，课件里面 Contraction property 的第二个公式有疏漏，不等号右边的算子 T 漏了 μ 下标，笔记里已经补上了。教授定义了收缩性表达式的新符号，定义 $||J|| = \max_x |J(x)|$ 并且叫它‘sup-norm’，但是实际上，$\max_x |J(x)|$ 在数学上是无穷范数 $||J||_\infty$ 的定义，所以 $||J||$ 的表达其实也不严谨，在这之后的笔记中出现了‘sup-norm’都不加说明地换成 $||J||_\infty$。使用无穷范数可以简化收缩性的表达，如下：

$$||TJ - TJ'||_\infty \leq ||J - J'||_\infty$$
$$||T_\mu J - T_\mu J'||_\infty \leq ||J - J'||_\infty$$

单调性 (Monotonicity property)：
假设存在单调映射 T，则对于任意在状态空间 X 内的满足 $J \leq J'$ 的点，对于所有 $x \in X$ 与策略 μ 算子 T 都满足 $(TJ)(x) \leq (TJ')(x)$ 与 $(T_\mu J)(x) \leq (T_\mu J')(x)$

压缩性 (Contraction property)：
假设算子 T 与 T_μ 具有压缩性，则对于任意有界函数 J 与 J' 都有：

$$\max_x |(TJ)(x) - TJ'(x)| \leq \alpha \max_x |J(x) - J'(x)|$$
$$\max_x |(T_\mu J)(x) - T_\mu J'(x)| \leq \alpha \max_x |J(x) - J'(x)|$$

动态规划算子的压缩性前面已经证明过了，单调性没有证明，目前只知道与泛函的知识有关，笔者缺乏相关知识，目前没法给出证明，暂时留着，等看看泛函再说。

Compact Contraction Notation：
这里的范数定义不太准确，和笔者查到的定义是不一样的，课件中给出的定义应该是无穷范数 $||J||_\infty = \max_x |J|$，但是教授给出的定义符号是 $||J||$，就按照课件的定义来。假设 T 是一个压缩映射，那么对于任意 x 与 μ，T 与 T_μ 具有性质：

$$||TJ - TJ'|| \leq \alpha ||J - J'||$$
$$||T_\mu J - T_\mu J'|| \leq \alpha ||J - J'||$$

再重复一次，我们讨论的折扣问题，算子是压缩映射。

3.5 THE TWO MAIN ALGORITHMS: VI AND PI

动态规划最基本也最重要的两种算法：值迭代和策略迭代，这个课程主要关心无限期折扣问题，所以值迭代和策略迭代给的都是无限期折扣问题的 bellman 方程，收敛性和最优性都在 lecture 1 中给出了证明，这里就不重复了，下面要分析一下这些 bellman 方程的求解情况。

假设这是一个离散状态离散控制的问题，有 n 个状态，相应地，就有 n 个 $J(x)$ 值，即 $J(x)$ 是一个输入向量输出向量的函数，按照 bellman 方程，就有一个 $n \times n$ 的方程组，同时由于 bellman 方程做的是期望计算，期望是一个线性操作，所以这是一个 $n \times n$ 的线性方程组。对于给定的策略 μ，方程组可以表示为 $\overrightarrow{J_\mu} = \overrightarrow{g_\mu} + \alpha \mathbf{P}_\mu \overrightarrow{J_\mu}$，理论上可以通过 $\overrightarrow{J_\mu} = -\overrightarrow{g_\mu}(\alpha \mathbf{P}_\mu - \mathbf{I})^{-1}$，式中的 $\mathbf{I}$ 是单位矩阵。这种方法是有局限性的，使用计算机进行矩阵的逆运算难度非常大，在实际解决问题的时候系统状态空间非常大，可能有数以万亿或者无数个状态，这种问题根本没有办法用计算机直接求解 $\overrightarrow{J_\mu}$ 的值，所以没有办法直接使用策略迭代和值迭代，必须做一些近似才能用策略迭代和值迭代求解。

现在需要做的就是看一下为什么策略迭代能够工作，也就是策略改进，根据当前策略 μ 计算出了相应的成本函数，求解策略改进表达式得到更好的策略可以保证策略不会恶化，所以每一次迭代都可以让策略均匀地得到改善。以笔者的经验，精确地计算策略成本函数值可以保证策略不会恶化，但是近似方法计算策略成本函数并不能保证策略不会恶化，这时候就需要分析近似方法计算出的策略成本函数能不能保证策略不恶化了。

3.6 JUSTIFICATION OF POLICY ITERATION

这里对策略迭代的最优性做一下证明：

Proof $J_{\mu^k} \geq J_{\mu^{k+1}}$:

对于给定的策略迭代次数 k，有定义 $J_{\mu^k} = T_{\mu^k} J_{\mu^k}$，根据策略改进的定义 $TJ_{\mu^k} = T_{\mu^{k+1}} J_{\mu^k}$，由于 T 与 $T_{\mu^{k+1}}$ 都基于 J_{μ^k} 计算，而且 T 是 J_{μ^k} 下策略空间中的最优策略，$T_{\mu^{k+1}}$ 是 J_{μ^k} 下策略空间中任意一个策略，所以有 $T_{\mu^k} J_{\mu^k} \geq TJ_{\mu^k}$，所以有 $J_{\mu^k} = T_{\mu^k} J_{\mu^k} \geq TJ_{\mu^k} = T_{\mu^{k+1}} J_{\mu^k}$，此时我们获得了两个已知大小关系 $J_{\mu^k} \geq T_{\mu^{k+1}} J_{\mu^k}$ 的点 J_{μ^k} 和 $T_{\mu^{k+1}} J_{\mu^k}$，根据单调性，可以得到：$J_{\mu^k} \geq T_{\mu^{k+1}} J_{\mu^k} \geq T^2_{\mu^{k+1}} J_{\mu^k} \geq \cdots \geq \lim_{N \to \infty} T^N_{\mu^{k+1}} J_{\mu^k}$。由于策略评价的迭代式定义为：$\lim_{N \to \infty} T^N_{\mu^{k+1}} J_{\mu^k} = J_{\mu^{k+1}}$，所以可以得到 $J_{\mu^k} \geq J_{\mu^{k+1}}$。

证明完毕

如果迭代过程中出现了 $J_{\mu^k} = J_{\mu^{k+1}}$，即 $J_{\mu^{k+1}}$ 为 bellman 方程的解，迭代即可停止，此时得到了动态规划的最优解 $J_{\mu^k} = J^*$。由此可知，策略迭代要么对策略进行严格的改进 (即 $J_{\mu^k} > J_{\mu^{k+1}}$)，要么就找到了最优解 (无法继续改进策略的时候)。这个结论在任意状态空间内都成立，如果 MDP 状态空间有限，那么经过有限步迭代后一定会终止于最优策略，原因是状态空间有限，控制空间也有限的时候，策略也是有限的，每次迭代都能够严格地对策略进行改进，最坏情况是遍历策略空间，也可以经过有限次迭代完成计算。对于无限状态空间的问题，策略迭代过程中可以看到解在逐渐收敛，只要经过足够多次的迭代，就可以达到你想要的精度。

我们讲过了值迭代和策略迭代，进行价值计算的时候都要迭代到收敛才能得到策略对应的价值或最优的价值，实际上策略迭代过程中进行少量策略

评价迭代来计算策略价值，然后使用这个不精确的策略价值进行策略改进也可以得到一个近似的策略，这种方法被叫做乐观策略迭代 (optimistic policy iteration)，有很多算法都使用了这个思想。

3.7 OPTIMISTIC POLICY ITERATION

策略迭代的策略评价算法标准情况下是迭代直到收敛的，但是 Optimistic PI 不会迭代到收敛，而是迭代若干次 (m 次)，不等到收敛就停止，然后进行策略改进，继续做策略评价，这也是一种近似方法。有整数 $m \in [1, \infty)$，则策略评价有 $J_{\mu^k} \approx T_\mu^m J$。对于给定的整数 m_k，有策略改进 $T_{\mu^k} J_k = T J_k$ 和策略评价 $J_{k+1} = T_{\mu^k}^{m_k} J_k$，$k = 0, 1, \cdots$。如果 $m_k \equiv 1$，算法使用最优算子算出 J_k 下的最优策略然后计算相应的 J_{K+1}，算法变成了值迭代，如果 $m_k = \infty$，算法使用最优算子算出 J_k 下的最优策略，然后迭代求该策略下的策略成本的不动点，此时算法就是策略迭代了。对于折扣问题，无论状态空间是有限的还是无限的，都能够保证收敛 (无限空间的问题进行无限次迭代也可以保证收敛)。

需要指出的是，对于大规模问题的时候，Optimistic PI 比 VI 和 PI 速度都快。

3.8 APPROXIMATE PI

近似策略迭代算法，包括近似策略评价和近似策略改进，计算当前策略 μ^k 下的策略成本 J_{μ^k} 时，使用近似方法计算近似成本 J_k 来代替 J_{μ^k}，近似方法可以保证误差上界：$||J_k - J_{\mu^k}||_\infty \leq \delta, \quad k = 0, 1, \cdots$，这里面的 $\delta > 0$ 没有给出更详细的解释，笔者的理解是 δ 有两种解释，一种是直接参与近似策略评价的过程，一种是作为近似策略评价迭代停止的条件，总之现在有近似策略评价误差上界 δ。

近似策略改进与近似策略评价相似，不管是直接参与近似策略改进还是作为近似策略改进的停止条件，有近似策略改进的误差上界 $\epsilon > 0$ 满足误差上界不等式：$||T_{\mu^{k+1}} J_k - T J_k||_\infty \leq \epsilon, \quad k = 0, 1, \cdots$，使用近似策略改进算子 $T_{\mu^{k+1}}$ 算出新的策略 μ^{k+1}，然后再对 μ^{k+1} 进行策略评价，直到近似策略迭代算法终止。

近似策略迭代过程中给出了策略 $\left\{\mu^k\right\}$，此时的策略成本也是有上界的，给出近似策略迭代过程中的近似成本与最优成本的误差上界：

$$\lim_{k \to \infty} ||J_{\mu^k} - J^*||_\infty \leq \frac{\epsilon + 2\alpha\delta}{(1-\alpha)^2}$$

证明如下：

Proof $\lim_{k \to \infty} ||J_{\mu^k} - J^*||_\infty \leq \frac{\epsilon + 2\alpha\delta}{(1-\alpha)^2}$:

证明完毕

α 是折扣问题中的折扣因子，这个折扣因子需要人为调整，如果折扣因子特别接近一，分母就会非常小导致误差很大，但是折扣因子非常小，趋近于零的时候，bellman 方程就会很接近于贪心算子。

近似策略迭代有一个非常典型的表现，就是随着迭代进行策略与成本稳定到某个点后开始在稳定的点附近震荡。这个时候不能说算法收敛了，算法会导致解在某个区域内震荡，这个区域不是特别大，而且由成本的误差上界决定。这是一种非常典型的表现，笔者在实验过程中经常会遇到这种现象，即目标值在某个值附近震荡。

近似策略迭代的另一个界，是策略稳定后产生的，上面的界是迭代过程中产生的，这个界是迭代终止，比如程序不停地生成相同的策略 $\bar{\mu}$，这时候就可以终止迭代了，此时成本函数有上界：$||J_{\bar{\mu}} - J^*||_\infty \leq \frac{\epsilon+2\alpha\delta}{1-\alpha}$，课件中没有给出计算，笔者也不知道如何证明，暂时不给出推导与证明了。

近似策略评价的表现不稳定，有些方法可以保证迭代终止并可以保证能够获得更好的误差，但是有些方法没法保证迭代终止，甚至没办法保证震荡终止。这两个界主要作用是给出一些参照，并不是决定性的指标，同时在做近似策略评估的时候没法一开始就知道 δ 的值到底是多少比较合适，需要尝试，同样，近似策略改进的 ϵ 的值也需要进行调整。

3.9 Q-FACTORS I

之前定义成本函数 $J(x)$ 的输入是系统状态 x，现在给出另一个成本函数的定义 $Q(x,u)$，这个函数的输入是系统状态 x 和控制 u，Q 函数定义如下：$Q^*(x,u) = E\{g(x,u,w) + \alpha J^*(\bar{x})\}$，$\bar{x} = f(x,u,w)$。定义为当前系统状态 x 下执行控制 u 产生的即时成本与新系统状态 $\bar{x}$ 开始，一直采取最优控制产生的累积成本的和的期望。Q 函数与 J 函数的关系为：$J^*(x) = \min_{u\in U(x)} Q^*(x,u)$，$\forall x$。$Q$ 函数执行当前控制后后续一直采取最优控制，J 函数从当前开始的所有控制都采取最优控制。

此时有值迭代的 J 函数 bellman 迭代方程：$J_{k+1}(x) = \min_{u\in U(x)} Q_{k+1}(x,u)$，$\forall x$

与 Q 函数 bellman 迭代方程：$Q_{k+1}(x,u) = E\left\{g(x,u,w) + \alpha \min_{v\in U(\bar{x})} Q_k(f(x,u,w),v)\right\}$。

Q 函数的值迭代算法就可以使用上面的迭代式计算了，值迭代结束之后就可以直接根据给定的状态 x 选择 Q 值最小的控制 u 了。

3.10 Q-FACTORS II

Q 函数的另一种解释是重建 MDP 模型，新的系统状态变为原模型的状态与控制，使用 Q 函数表示成本函数，则原模型的 bellman 算子 T 变为 F，此时有 bellman 方程 $Q^* = FQ^*$，详细的表达式：

$$(FQ)(x,u) = E\left\{ g(x,u,w) + \alpha \min_{v \in U(\bar{x})} Q(\bar{x},v) \right\}, \quad \bar{x} = f(x,u,w)$$

在数学上，同一个问题使用 Q 函数与 J 函数表示成本函数进行精确值迭代与策略迭代都是一样的，无论要进行的操作还是获得的结果都完全一样，不一样的地方是 Q 函数要维护的向量要比 J 函数更多，需要的存储空间更大，更大的区别发生在近似方法 (近似值迭代与近似策略迭代) 求解过程中，最主要的区别是，如果我已经计算得到了精确的 Q 值与 J 值，使用 J 值计算最优控制的时候需要计算当前状态转移到新状态得到的期望累加成本，需要知道当前状态下执行控制转移到新状态的概率，也就是说 J 函数必须需要系统模型才能使用。而 Q 函数得到当前状态 x 时，计算最优控制只需要在 x 下找到最小的 Q 值对应的控制就可以了，也就是说 Q 函数不需要系统模型也可以使用。

在不知道系统模型的时候，J 函数是无法使用的，而 Q 函数就不受影响，当然计算 Q 值的时候还是需要一个模拟器，从模拟器中获得数据然后计算 Q 值，使用模拟器也是一种近似思路，在后面会介绍。

3.11 OTHER DP MODELS

我们之前讨论的有限期与无限期/连续或离散折扣问题分析起来更简单，也有更强的理论支撑我们来看一看其他动态规划模型：

1. **非折扣问题 ($\alpha = 1$)**：这个非折扣问题包括两种：有限期问题与无限期问题，有限期问题比较直观，成本函数是有上界的 (如果每阶段的成本都有上界)。无限期问题有两种，第一种是系统状态跳转没有终止，这时候没有折扣，$\alpha = 1$，只有一种情况成本函数有界，即经过有限步跳转之后即时成本收敛到 0，比如使用电机控制水泵调节液位高度，当液位到达给定高度后，电机不需要消耗很多能量就可以调节液位高度，这是成本函数即消耗的能量是趋于零的。第二种无限期问题是展望期不固定的问题，比如无向图或者有向图中单源点单汇点的随机最短路径问题，在某节点选择新节点后不能保证确定地到达这个新节点，而是以一个概率分布到达新的节点位置，这种问题的路径长度与转移次数不固定，也被归为无限期问题，这一类问题的成本函数被认为是有界的。
2. **连续时间有限状态的 MDP**：这里先区分一下连续时间与离散时

间的马尔科夫链，离散时间指的是每经过相同的时间间隔系统状态就跳转一次的马尔可夫链，连续时间是相邻两个状态产生的时间间隔不是均匀的，任意时刻都可能有新状态产生，以笔者所知，有两种连续时间，第一种是每时每刻都在产生新状态，即状态 $x(t)$ 是关于时间 t 的函数，比如电机控制等系统，另一种是两个状态以满足某概率分布的时间间隔进行跳转，一个比较典型的例子，是排队系统，一个客户 (可能以泊松分布等概率分布) 到达，状态增加 1，客户以随机时间离开，随机的时间长短依赖于控制 (服务器数量)，实际上，排队系统详细划分应该属于半马尔科夫链 (semi-markov decision process, semi-MDP)

这里要特别详细讲一下连续时间和连续空间的模型，即状态空间与控制空间都连续的模型。这是一类典型的控制问题，比如控制电机的电流调节水箱液位，控制飞机飞行，车辆行驶，机器人完成任务与一些过程，比如化学生产过程等，这类问题关注时间，状态和控制都连续的问题，从数学上讲，这类问题更复杂，特别是有随机干扰与不确定因素的时候。即使是确定性的连续时间问题也很难解决，这需要解决一个与微分方程相关的问题，需要非常强的理论支撑，实际上一部分的连续时间系统可以通过离散化手段来分析。

3.12 CONTINUOUS-TIME MODELS

连续时间模型的 Hamilton-Jacobi-Bellman(HJB) 方程有很多推导方式，这里使用课件的方法进行推导。

有一个系统，系统方程为 $dx(t)/dt = f(x(t), u(t))$，成本函数为 $\int_0^\infty g(x(t), u(t))$，使用 $J^*(x)$ 表示最优成本。使用 δ 将连续时间系统离散化为 $x_{k+1} = x_k + \delta f(x_k, u(k))$，此时离散化后的最优成本计算表达式为:

$$J_\delta^*(x) = \min_u \{\delta g(x, u) + J_\delta^*(x + \delta f(x, u))\}$$

对上式左右两边同时除以 δ 可以得到：

$$\begin{aligned}
&\frac{J^*_\delta(x)}{\delta} = \frac{\min_u \{\delta g(x,u) + J^*_\delta(x+\delta f(x,u))\}}{\delta} \\
\Rightarrow &\frac{J^*_\delta(x)}{\delta} = \min_u \left\{ \frac{\delta g(x,u) + J^*_\delta(x+\delta f(x,u))}{\delta} \right\} \\
\Rightarrow &\frac{J^*_\delta(x)}{\delta} = \min_u \left\{ g(x,u) + \frac{J^*_\delta(x+\delta f(x,u))}{\delta} \right\} \\
\Rightarrow &\frac{J^*_\delta(x)}{\delta} = \min_u \left\{ g(x,u) + \frac{J^*_\delta(x+\delta f(x,u)) - J^*_\delta(x) + J^*_\delta(x)}{\delta f(x,u)} f(x,u) \right\} \\
\Rightarrow &\frac{J^*_\delta(x)}{\delta} = \min_u \left\{ g(x,u) + \bigtriangledown_x J^*_\delta(x)' f(x,u) + \frac{J^*_\delta(x)}{\delta f(x,u)} f(x,u) \right\} \\
\Rightarrow &\frac{J^*_\delta(x)}{\delta} = \min_u \left\{ g(x,u) + \bigtriangledown_x J^*_\delta(x)' f(x,u) + \frac{J^*_\delta(x)}{\delta} \right\} \\
\Rightarrow &\frac{J^*_\delta(x)}{\delta} = \min_u \{ g(x,u) + \bigtriangledown_x J^*_\delta(x)' f(x,u) \} + \frac{J^*_\delta(x)}{\delta} \\
\Rightarrow &0 = \min_u \{ g(x,u) + \bigtriangledown_x J^*_\delta(x)' f(x,u) \}
\end{aligned}$$

由于 $\lim_{\delta\to 0} \{\bigtriangledown_x J^*_\delta(x)'\} = \bigtriangledown_x J^*(x)'$，同时对等号左右两边取极限 $\delta \to 0$

$$\begin{aligned}
0 &= \lim_{\delta\to 0} \min_u \{ g(x,u) + \bigtriangledown_x J^*_\delta(x)' f(x,u) \} \\
\Rightarrow 0 &= \min_u \left\{ g(x,u) + \lim_{\delta\to 0} \{ \bigtriangledown_x J^*_\delta(x)' f(x,u) \} \right\} \\
\Rightarrow 0 &= \min_u \{ g(x,u) + \bigtriangledown_x J^*(x)' f(x,u) \}
\end{aligned}$$

在策略迭代中，求解 $0 = g(x,\mu(x)) + \bigtriangledown_x J^*(x)' f(x,\mu(x))$ 直到收敛，然后使用 $\bar{\mu}(x) \in \arg\min_u \{ g(x,u) + \bigtriangledown_x J^*(x)' f(x,u) \}$ 对策略进行改进。

这种方法需要计算 $J^*(x)$ 的梯度，有一个问题就是 $J^*(x)$ 必须有梯度，否则无法计算，同时算法学习的是 $\bigtriangledown_x J^*(x)'$ 而不是 $J^*(x)$，离散时间问题与连续时间问题在这个地方是不一样的，这个很重要。

3.13 A MORE GENERAL/ABSTRACT VIEW OF DP

给出一个更一般更抽象的动态规划，任意的实空间 Y，有一个映射 $F: Y \mapsto Y$，当存在 $\rho \in (0,1)$ 满足 $||Fy - Fz|| \leq ||y - z||, \forall y, z \in Y$ 时称 F 是压缩映射，ρ 被叫做映射 F 的压缩模量。

有集合 X 与映射 $v: X \mapsto \Re$，而且 v 的输出是一个正数，令 $B(X)$ 是所有映射 $J: X \mapsto \Re$ 的集合，则 $\frac{J(x)}{v(x)}$ 是一个有界集合。

定义集合 $B(X)$ 中的加权最大范数 (weighted sup-norm)：$||J|| = \max_{x\in X} \frac{|J(x)|}{v(x)}$，那么动态规划折扣问题的映射 T 和 T_μ 就是 $v(x) \equiv 1, \rho = \alpha$ 的加权最大范数。说明一件事情，后面的内容中如果出现了 $||\cdot||$，指的是这里的定义，如果想要表达一般数学意义上的范。

3.14 CONTRACTION MAPPINGS: AN EXAMPLE

考虑一个状态空间无限大但状态可计量的动态规划问题，状态集合 $X = \{1, 2, \cdots\}$, 比如一个排队系统, 系统状态为客户数量, 客户数量可能是 $1, 2, \cdots$, 每个阶段的成本依赖于系统状态而且没有上界，之前的，之前的分析可以知道这种系统很难求解，现在要给出一个让成本函数有界的方法，即使用加权最大范数 (sup-weighted norm) 处理这个问题让它有界。

映射 T_μ 有成本函数 $(T_\mu J)(i) = b_i + \alpha \sum_{j\in X} a_{ij} J(j), \forall i = 1, 2, \cdots$，其中 i 和 j 表示系统状态，b_i 和 a_{ij} 是两个标量，b_i 表示即时成本，a_{ij} 表示从状态 i 到状态 j 的转移概率，$v(i)$ 表示每个阶段状态 i 下的成本，当且仅当存在 $\rho \in (0, 1)$ 满足 $\frac{\sum_{j\in X} |a_{ij}|v(j)}{v(i)} \le \rho, \forall i = 1, 2, \cdots$ 时，T_μ 是压缩模量为 ρ 的压缩映射。对于映射 T 有 $(TJ)(i) = \min_\mu (T_\mu J)(i), \forall i = i, 1, \cdots$，如果所有策略 $\mu \in M$，T_μ 都是以 ρ 为压缩模量的压缩映射，则映射 T 也是以 ρ 为压缩模量的压缩映射。

这种方法让动态规划的适用范围从每阶段成本函数有界问题扩展到了每阶段成本函数无界问题。

实际上，这一页的介绍有一点不理解的地方，就是 $\frac{\sum_{j\in X} |a_{ij}|v(j)}{v(i)} \le \rho, \forall i = 1, 2, \cdots$ 中 $v(i)$ 的实际意义，从 bellman 公式中可以知道每阶段即时成本为 b_i，但是这里又说每阶段成本是 $v(i)$，没弄明白这个 $v(i)$ 指的到底是什么。

3.15 CONTRACTION MAPPING FIXED-POINT TH.

本页给了点压缩映射不动点的理论。

压缩映射不动点定理：如果映射 $F : B(X) \mapsto B(X)$ 是一个以 $\rho \in (0, 1)$ 为压缩模量的映射，一定存在一个唯一的 $J^* \in B(X)$ 满足 $J^* = FJ^*$，更一般地，任意 $J \in B(X)$，都有 $\{F^k J\}$ 收敛于不动点 J^*，同时有 $||F^k J - J^*|| \le \rho^k ||J - J^*||, k = 1, 2, \cdots$。

另一个理论是：如果一个范数向量空间 Y 中所有柯西序列 $\{y_k\}$ 都收敛，那么这个空间 Y 是一个完备空间 (这个性质在上面动态规划收敛且最优的证明的部分提到过)。

特殊地，空间 $B(X)$ 是完备空间。

3.16 ABSTRACT FORMS OF DP

本页给出了满足单调性和压缩性的动态规划的抽象形式。

映射 $J : X \mapsto \Re$ 表示从状态空间 X 到实标量空间 $\Re$ 的映射，所有映射构成了一个映射空间 $R(X)$，现在定义映射 $H : X \times U \times R(X) \mapsto \Re$，按照之前给出的 bellman 方程的定义，可以使用 H 改写 mellman 方程：$(TJ)(x) = \min_{u \in U(x)} H(x, u, J), \forall x \in X$，这样做扩大了空间，但是可以更间接地表达 bellman 方程与求解过程，不需要写出整个的表达式了。

现在假设 $(TJ)(x) > -\infty, \forall x \in X$，就可以得到映射 T 是从空间 $R(X)$ 到空间 $R(X)$ 的映射。

令 M 表示所有策略 μ 的集合，则对于所有 $\mu \in M$，T_μ 与 H 满足关系：$(T_\mu J)(x) = H(x, \mu(x), J), \forall x \in X$

如果能找到一个不动点 J^*，则存在

$$J^* = (TJ^*) = \min_{u \in U(x)} H(x, u, J^*), \forall x \in X$$

3.17 EXAMPLES

举几个例子来说明这种表达方式是如何描述不同的问题的：

1. **折扣问题**：

$$H(x, u, J) = E\left\{g(x, u, w) + \alpha J(f(x, u, w))\right\}$$

2. **折扣的离散状态连续时间的半马尔科夫问题 (比如排队系统)**：

$$H(x, u, J) = G(x, u) + \sum_{y=1}^{n} m_{xy}(u) J(y)$$

   其中 m_{xy} 就是折扣因子 (discounted) 也就是转移概率，一个通过转移时间定义的分布，从这个例子中可以得到，折扣因子可以有实际的物理意义，不一定是为了构造有界成本函数给出的衰减因子 $\alpha \in (0, 1)$。

3. **极大极小 (minimax) 问题/博弈问题**：

$$H(x, u, J) = \max_{w \in W(x,u)} \left[g(x, u, w) + \alpha J(f(x, u, w))\right]$$

   主要应用于回合制游戏或者回合制决策问题，式中的 $w \in W(x, u)$ 不是随机变量，而是对手做出的决策，这个决策依赖于当前系统状

态与你做出的决策，你在状态 x 时做出了决策 u，然后你的对手跟你决策时的 x 和 u 做出了一个相应的决策，然后你观察系统的状态可以得到一个新的 x，再继续做决策，然后对手观察，决策，这么循环直到游戏结束。这种模式有点像鲁棒优化或者鲁棒控制，即在最坏的情况下给出最好的方案，因此不是期望而是一个确定性的指标进行评价，几乎所有有对手的对抗性游戏都是这一类型的。

4. **最短路径问题**：现在有一个 n 个节点的图，知道目标节点，每一次从一个节点到另一个节点，同时产生相应的转移成本，想要找到从任意一个节点开始转移到终点的成本最小的路径。可以把它看作一个从状态 x 开始的动态规划问题，图的节点是你在当前节点选择的控制 u，然后就可以从当前节点到选择的节点 u。J 是从图上不同节点开始走直到终点的成本函数，所以映射 H 是从当前节点直到终端节点的成本函数的映射，需要注意的是终端节点的成本是零，这就是一个确定性的最短路径问题。如果你选择了一个节点，但是新的节点是根据概率分布跳转产生的，表现为 w，这就是一个随机最短路径问题，如果从 minimax 角度出发获考虑这个问题，也就是你做出决策后新节点是被对手选择的，就不需要使用折扣因子了。

总之这是一个通用性很强的框架，我们可以使用这个统一的理论处理所有相似的问题，只要简单地修改一下 H 的定义就可以了。

3.18 ASSUMPTIONS

对于 H 算子有一些假设，下面说一说这些假设和假设带来的性质：

1. **单调性**：假设映射 H 具有单调性，当存在 $J, J' \in R(X)$ 并且 $J \leq J'$，有 $H(x,u,J) \leq H(x,u,J'), \forall x \in X, u \in U(x)$

   如果映射具有单调性，那么所有的动态规划问题都可以使用相同的标准分析过程和计算结果。

2. **收缩性**：如果对于所有 $J \in B(X)$，映射结果 $T_\mu J$ 和 TJ 都在 $B(X)$ 中，那么一定存在 $\alpha \in (0,1)$，对于所有策略 $\mu \in M$ 和 $J, J' \in B(X)$ 满足 $||T_\mu J - T_\mu J'|| \leq \alpha ||J - J'||$

   这个性质存在的意义在于如果 H 具有压缩性，那么可以知道 T 和 T_μ 也具有压缩性。

3. **只有单调性 (比如非折扣问题)**：只有一部分结果可以用。

4. **部分压缩性**：如果一个动态规划问题，对于某些策略 μT_μ 具有压缩性，但是其他的 μT_μ 就不具有压缩性，这种性质叫做半压缩性 (semi-contractivenss)，或者叫非压缩 (noncontractive)，这时候由于算子没法保证压缩性，也就没法保证最优性，所以对于这种问题，一定要想办法让算子具有压缩性。

3.19 RESULTS USING CONTRACTION

满足压缩性后能获得的几个结果，证明暂时略过：

Proposition 不动点定理：
如果映射 T_μ 和 T 都是 $B(X)$ 空间内以 α 为范数模量的加权最大范数压缩映射 *(weighted sup-norm contraction mapping with modulus α)*，一定会分别存在唯一的不动点 J_μ 和 J^*

Proposition 值迭代收敛性： 如果算子 T_μ 和 T 具有压缩性，那么对于任意成本函数 $J \in B(X)$ 和策略 $\mu \in M$，存在 $\lim_{k\to\infty} T_\mu^k J = J_\mu$ 和 $\lim_{k\to\infty} T^k J = J^*$

Proposition 策略迭代最优性： 当且仅当 $J_\mu = J^*$ 时，有 $T_\mu J^* = TJ^*$，即最优策略。

3.20 RESULTS USING MON. AND CONTRACTION

单调性和收缩性的几个结论：

对于任意的 $\epsilon > 0$，一定存在策略 $\mu_\epsilon \in M$ 满足 $J^x \le J_{\mu_\epsilon} \le J^*(x) + \epsilon, \forall x \in X$

有所有策略 $\pi = \{\mu_0, \mu_1, \cdots\}, \forall \mu_k \in M$ 的集合 Π，定义

$$J_\pi(x) = \min_{k\to\infty} \inf (T_{\mu_0} T_{\mu_1} \cdots T_{\mu_k})(x), \forall x \in X$$

满足 $J^*(x) = \min_{\pi\in\Pi} J_\pi(x), \forall x \in X$，这时就不是平稳策略的最优解了，而是包括了非平稳策略，在整个策略空间内的最优解。

3.21 THE TWO MAIN ALGORITHMS: VI AND PI

现在要用一个更一般性的表达方式来描述值迭代与策略迭代算法，值迭代和策略迭代的细节就不描述了，下面直接给出他们的一般性表达形式，给定整数 m_k，有动态规划算法：

$$T_{\mu^k} J_k = TJ_k, \quad J_{k+1} = T_{\mu^k}^{m_k} J_k, \quad k = 0, 1, \cdots$$

1. 如果 $m_k \equiv 1$，这个算法就是值迭代
2. 如果 $m_k \equiv \infty$，这个算法就是策略迭代

3. 如果 $m_k \in (1,\infty), m_k \in \mathbb{Z}$，则策略评价执行若干次迭代，然后进行一次策略改进，这种方法比值迭代和策略迭代效率都高。

3.22 ASYNCHRONOUS ALGORITHMS

异步算法的优势主要有三个：收敛速度快、可以并行分布式计算、可以使用仿真来计算。

把状态集合 X 分成不相交的非空子集合 $X_1, X_2, \cdots, X_m$，相应的成本函数 J 被分为与状态子集合相同个数的成本函数集 $J = (J_1, J_2, \cdots, J_m)$，这样 J_ℓ 就和相应的 X_ℓ 一一对应了。

同步值迭代算法中，成本函数 J 的更新公式为：

$$J_\ell^{t+1}(x) = T(J_1^t, J_2^t, \cdots, J_m^t)(x), x \in X_\ell, \ell = 1, 2, \cdots, m$$

上式中 t 表示迭代次数，这个迭代表达式表示每次同时更新全部成本函数。

异步值迭代算法中，使用 ℓ 个计算机计算成本函数 J 的更新公式为：

$$J_\ell^{t+1}(x) = \begin{cases} T(J_1^{\tau_{\ell 1}(t)}, J_2^{\tau_{\ell 2}(t)}, \cdots, J_m^{\tau_{\ell m}(t)})(x), & t \in \mathcal{R}_\ell \\ J_\ell^t(x), & t \notin \mathcal{R}_\ell \end{cases}$$

上式中 $\mathcal{R}_\ell$ 为 ℓ 个表示时间的子集合，$J_\ell^{t+1}(x)$ 为第 ℓ 个成本函数集合第 $t+1$ 次迭代的结果，如果当前迭代次数 t 在时间子集合 $\mathcal{R}_\ell$ 中，则对第 $\tau_\ell(t)$ 次迭代时的成本函数 $J^{\tau_{\ell 1}(t)}$ 使用算子 T，即将 $T(J_1^{\tau_{\ell 1}(t)}, J_2^{\tau_{\ell 2}(t)}, \cdots, J_m^{\tau_{\ell m}(t)})(x)$ 的值赋给 $J_\ell^{t+1}(x)$，否则不做任何操作，直接把时间 t 的成本函数 $J_\ell^t(x)$ 赋给成本函数 $J_\ell^{t+1}(x)$。$t - \tau_{\ell_j}(t)$ 即为改异步算法的通信延迟，这种做法每次迭代时不同时更新所有成本函数，只更新一部分，更新时不对当前成本函数使用算子，而是对若干次迭代前的成本函数使用算子。这也是同步算法与异步算法之间最大的区别，同步算法需要知道所有迭代信息，而异步算法可以不知道所有迭代信息。

3.23 ONE-STATE-AT-A-TIME ITERATIONS

一个比较重要的例子，假设有 n 个状态，分成了 n 个子集，使用仿真方法生成系统状态，每一个状态都会产生无数次，使用 n 个机器去计算 $J(x)$ 的值，这种情况下算法是没有通信延迟的。

有异步值迭代算法迭代表达式：

$$J_\ell^{t+1}(x) = \begin{cases} T(J_1^t, J_2^t, \cdots, J_n^t)(\ell), & \ell = x_t \\ J_\ell^t, & \ell \neq x_t \end{cases}$$

上式中 $T(J_1^t, J_2^t, \cdots, J_n^t)(\ell) = TJ^t$，其中 $T(J_1^t, J_2^t, \cdots, J_n^t)(\ell)$ 表示成本向量第 ℓ 个元素。

如果系统状态序列为 $\{x^0, x^1, \cdots\} = \{1, 2, \cdots, n, 1, 2, \cdots, n, 1, 2, \cdots\}$，这样迭代时就先访问第一个状态，更新第一个状态的成本，然后访问第二个状态的成本，一次访问下去直到最后一个状态 x^n 的成本，再从第一个状态重新开始直到收敛，这种迭代方法很经典，叫做高斯-赛德尔迭代 (Gauss-Seidel method)

3.24 ASYNCHRONOUS CONV. THEOREM I

异步值迭代与异步策略迭代需要对普通版本的值迭代和策略迭代做一点修改才能让异步算法工作，这个修改是建立在两个假设的基础上，第一个假设是每一个机器都能够进行无限次数的更新，即每个状态都会被访问无数次，也就是课件中给的 $\Re_\ell \to \infty$ 想要表达的假设，第二个假设是延迟通信会一直进行下去，算法会一直传递 (延迟的) 信息，τ 与 t 永远会存在一个 gap，即课件中 $\lim_{t\to\infty} \tau_{\ell_j}(t) = \infty$ 要表达的内容。这就是两个比较基本的假设。

如果存在非空序列 $\{S(k)\} \subset R(X)$ 同时满足 $S(k+1) \subset S(k), \forall k$ 并且算子 T 有唯一的不动点 J^*，那么有如下两个性质:

1. **同步收敛条件**: 所有的序列 $\{J^k\}, J^k \in S(k)$，最终都会收敛于不动点 J^*，更多地，有 $TJ \in S(k+1), \forall J \in S(k), k = 0, 1, \cdots$
2. **Box Condition(不知道中文怎么翻译)**: 对于所有的 k，$S(k)$ 是 $S_i(k)$ 的笛卡儿积，即 $S(k) = S_1(k) \times S_2(k) \times \cdots S_m(k)$，式中 $S_\ell(k)$ 是集合 $X_\ell, \ell = 1, 2, \cdots, m$ 上的实函数集合。

可以知道，对于任意 $J \in S(0)$，异步算法产生的序列 $\{J^t\}$ 一定会收敛于唯一的不动点 J^*。

3.25 ASYNCHRONOUS CONV. THEOREM II

这一页要多写一些东西，把上面几页讲的异步算法按照笔者的理解整理一下。

异步算法的一般性框架是把状态集合 X 拆成不相交的非空子集 $X_1, X_2, \cdots, X_m$，相应的成本函数为 $J_1, J_2, \cdots, J_m$，使用 m 个机器更新每一个子集 (X_ℓ, J_ℓ)。举个例子，三维离散状态空间

$$X = \{x_0, x_1, x_2 | x_0, x_1, x_2 \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}\}$$

这样的状态空间一共有 9^3 个状态，分成三个 3×9^2 个元素的子集

$$X_0 = \{(x_0, x_1, x_2) | x_0 \in \{1, 2, 3\}, x_1, x_2 \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}\}$$
$$X_1 = \{(x_0, x_1, x_2) | x_0 \in \{4, 5, 6\}, x_1, x_2 \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}\}$$
$$X_2 = \{(x_0, x_1, x_2) | x_0 \in \{7, 8, 9\}, x_1, x_2 \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}\}$$

相应地，有三个 $\Re^{3\times 9^2}$ 维的子成本向量函数 J_0, J_1, J_2，放到一起就是没有拆开的成本向量函数 $J = (J_0, J_1, J_2)$。

同步值迭代和异步值迭代，主要区别是同步值迭代每次迭代都一次性更新所有 J 的值，异步值迭代只有根据集合 $\mathcal{R}_\ell$ 在特定迭代次数的时候更新特定子集的 J_ℓ^{t+1}，有些时候异步值迭代有通信延迟，有些时候异步迭代没有通信延迟，这就是同步值迭代和异步值迭代的区别。

异步值迭代迭代过程中解 J 属于一个集合，如果异步值迭代算法能够收敛，那么 J 所在的集合就应该不断变小，同时每次算子作用在 J 上之后都要保证新的 J 还在当前集合中，课件中将 J 所在的集合拆开，以笛卡儿积的形式表示出来，这样也将 J 相应地拆开了，在迭代过程中，一个部分一个部分地优化，所有部分都优化过之后，J 就会往里面的集合里移动，随着迭代进行，集合越来越小，J 也距离 J^* 越来越近，直到最后收敛。所以课件中 p48 的两个性质是具有收敛性的两个必要条件，而不是充分条件，所以这个时候可以先证明收敛，然后使用这两个性质进行拆分，然后再进行迭代去求最优的 J。

Part III

APPROXIMATE DYNAMIC PROGRAMMING: OVERVIEW

4

LECTURE 3: OVERVIEW OF ADP - GENERAL ISSUES OF APPROXIMATION AND SIMULATION

CONTENTS

4.1 LECTURE OUTLINE

前两次课程讨论了了精确动态规划，重点介绍了折扣问题，Lecture 3 要先回顾一下折扣问题的精确动态规划，然后给出一个比较全面的综述，只是给一个大概的介绍，不会讲他们的细节，主要目的是介绍这个领域的状况，指出这个领域比较重要，需要重点关注的问题。

首先介绍近似动态规划，然后介绍近似结构，比如参数化近似形式和近似结构，然后会讲一下使用仿真的近似策略迭代，同时讨论为什么仿真技术在动态规划领域这么重要，接下来会讨论一些策略评估的近似方法和策略改进的问题，最后讲一些近似和方针的一般性的问题。

这次课程的主要目录如下：

1. Review of discounted DP
2. Introduction to approximate DP
3. Approximation architectures
4. Simulation-based approximate policy iteration
5. Approximate policy evaluation
6. Some general issues about approximation and simulation

4.2 DISCOUNTED PROBLEMS/BOUNDED COST

回忆一下折扣问题：一个离散时间下任意状态空间 x_k 内的平稳系统有状态方程 $x_{k+1} = f(x_k, u_k, w_k), k = 0, 1, \cdots$，新状态 x_{k+1} 受到旧状态 x_k、控制 u_k 与扰动/噪声 w_k 的影响。对于给定的策略 $\pi = \{\mu_0, \mu_1, \cdots\}$ 与初始状态 x_0，有成本函数 $J_\pi(x_0) = \lim_{N\to\infty} E\left\{\sum_{k=0}^{N-1} \alpha^k g(x_k, \mu_k, w_k)\right\}$，其中 $\alpha < 1$，同时对于所有 (x, u, w) 存在实数 M 满足 $|g(x, u, w) \le M|$，但是仅仅让每阶段的即时成本有界是不够的，如果我们要解决的是一个无限期问题，累加成本同样会有可能无界，这时候 α 就比较重要了，由于 $\alpha \in (0, 1)$，当 k 很大的时候 α^k 会非常小，这样就可以保证折扣成本序列收敛于 0，累加折扣成本有界。

定义动态规划最优算子 T：

$$(TJ)(x) = \min_{u\in U(x)} E\left\{g(x, u, w) + \alpha J(f(x, u, w))\right\}, \forall x$$

与动态规划策略 μ 的算子 T_μ：

$$(T_\mu J)(x) = E\left\{g(x, \mu(x), w) + \alpha J(f(x, \mu(x), w))\right\}, \forall x$$

这样我们就可以使用算子 T 和 T_μ 比较简单地表示动态规划 bellman 方程：$J = (TJ)(x)$ 与 $J = (T_\mu J)(x)$。

4.3 MDP - TRANSITION PROBABILITY NOTATION

这里给出一个用状态转移概率定义的动态规划模型，假设一个状态空间离散且有限的 (共 n 个状态) 马尔科夫链，系统状态使用 i 来表示 $(i = 1, 2, \cdots, n)$，这与之前我们表示状态的符号 x 不同，i 只表示离散可数状态空间里的状态，而 x 可以表示任意状态空间里的状态，状态转移概率 $p_{i_k,i_{k+1}}(u)$ 表示控制 u 下从状态 i_k 转移到状态 i_{k+1} 的概率，这时候状态转移概率就可以取代我们之前介绍的系统中的状态方程了。即时成本函数 $g(x_k, u_k, w_k)$ 也被新的即时成本函数 $g(i_k, u_k, i_{k+1})$ 代替，成本函数向量 J 表现为 $J = (J(1), J(2), \cdots, J(n))$，此时 J 为一个 n 维的实向量 $\Re^n$

策略 $\mu = \{\mu_0, \mu_1, \cdots\}$ 下的总成本定义为：

$$J_\mu(i) = \lim_{N\to\infty} E\left\{\sum_{k=0}^{N-1} \alpha^k g(i_k, u_k(i_k), i_{k+1}) | i_0 = i\right\}$$

该总成本表达式被概率转移 $p_{i_k,i_{k+1}}(u_k)$ 重新定义：

$$J_\mu(i) = \lim_{N\to\infty} \left\{\sum_{k=0}^{N-1} \prod_{t=0}^{t=k} p_{i_k,i_{k+1}}(u_k)\alpha^k g(i_k, u_k(i_k), i_{k+1}) | i_0 = i\right\}$$

为了便于表达，给出 bellman 方程的最优成本表达式与策略 $\pi = \{\mu_0, \mu_1, \cdots\}$ 成本表达式在状态转移下的算子 T 和 T_μ 下的定义：

$$(TJ)(i) = \min_{u\in U(i)} \sum_{j=1}^{n} p_{i,j}(u)(g(i, u, j), +\alpha J(j)), i = 1, 2, \cdots, n$$

$$(T_\mu J)(i) = \sum_{j=1}^{n} p_{i,j}(u)(g(i, \mu(i)), j), +\alpha J(j)), i = 1, 2, \cdots, n$$

然后就可以使用新定义下的算子 T 和 T_μ 表达 bellman 最优方程与 bellman 策略方程 $J(x) = (TJ)(x)$ 与 $J(x) = (T_\mu J)(x)$

4.4 "SHORTHAND" THEORY –A SUMMARY

这一页内容不多，很多内容之前已经出现很多次了，在这里只谈一件事情，就是动态规划的最优性条件：

$$\mu : optimal \Leftrightarrow T_\mu J^* = TJ^*$$

这个最优性条件可以解释为：如果对最优成本函数使用最优算子 T，得到的结果与对最优成本函数使用当前的策略算子 T_μ 得到的结果相等，那么当前

策略 μ 即为最优策略，反过来，已知某策略 μ 是最优策略，可以得到等式关系 $T_\mu J^* = TJ^*$，也就是最优性和那个条件是互为充要条件的。

但是这里面有一个问题，如果用条件 $T_\mu J^* = TJ^*$ 来判断当前策略是否最优的话，一定会失败，因为策略评价就是在使用这个表达式进行的，求解这个表达式得到的结果就是策略评价的结果，如果知道成本函数 J 是最优成本函数 J^* 的话，就可以确定满足最优性条件 $T_\mu J^* = TJ^*$，问题是迭代过程中没法知道当前成本函数是不是最优的，所以不能用这个条件判断是否最优，虽然他确实是最优成本函数。

但是使用条件 $T_\mu J^* = TJ^*$ 的话，就比较好判断，虽然这个条件遇上一个条件等价，但是能够判断是不是满足了最优条件，迭代过程中对当前的成本函数 J_μ 分别使用最优算子 T 和当前的策略算子 T_μ 得到 $(T_\mu J_\mu)$ 和 (TJ_μ)，如果 $(T_\mu J_\mu) = (TJ_\mu)$，就可以知道当前策略满足最优性条件了，当前策略即为最优策略。

这里要对这个最优性条件做一下分析，策略迭代过程中根据当前策略 μ 进行评估计算 J_μ，如果策略评价结束之后发现 $(T_\mu J_\mu) = (TJ_\mu)$，则当前策略 μ 一定是最优策略 μ^*，显然使用当前策略进行策略评价也就是使用最优策略进行策略评价，得到的成本函数很显然是最优成本函数 J^*，所以在观察到 $(T_\mu J_\mu) = (TJ_\mu)$ 的时候，实际上有 $(T_\mu J^*) = (TJ^*)$，也就是最优条件。

4.5 THE TWO MAIN ALGORITHMS: VI AND PI

动态规划有两个很基础也很重要的算法：值迭代和策略迭代，笔者不会在这一页中给出过多前面已经出现过的内容，包括公式，算法流程之类的东西，而是要在这一页做一些动态规划难以求解的原因的介绍。

bellman 方程表达式为：$J = (T'J)$，求解这个表达式需要不断地对当前的成本函数 J 使用算子 T'，这个 T' 指的是一般性的算子，一直迭代到 $J = (T'J)$，迭代就停止了，这时候成本函数 J 收敛到了最优成本函数，这就是迭代求解的过程。非迭代求解可以通过求解一个 bellman 方程组来完成，系统中有 n 个状态，成本函数会是一个 n 维的实向量，此时 bellman 方程组是一个 $n \times n$ 的线性方程组，求解这个方程组就可以得到最优的成本函数向量。如果 T' 使用最优算子 T，这个算法就叫做值迭代，如果 T' 使用策略算子 T_μ，那么这个算法就需要策略迭代，得到 J_μ^* 后，还需要使用 $T_\mu J_\mu = TJ_\mu$ 对策略进行改进，然后做策略评价-策略改进的循环直到收敛，同样的，策略改进需要求解 n 个最小化问题，也就是使用 bellman 最优方程计算最优控制 u，每一个最小化问题都需要在整个控制空间中搜索，求解难度同样很大。

所以动态规划需要处理的问题有两个：求解一个 $n \times n$ 的线性方程组和执行 n 次最小化操作，当 n 的值非常大的时候，不管求解方程组还是最小化都及其困难，控制空间同样会影响动态规划的求解难度，所以动态规划的维数灾是由状态空间和控制空间共同导致的，精确动态规划在解决问题时根本无法使用，所以需要使用近似方法来解决这些动态规划无法求解的问题。

4.6 GENERAL ORIENTATION TO ADP

80 年代末期出现了 ADP 的概念，这是一个能够解决维数灾问题让动态规划在大规模状态甚至是状态数量无限多的实例中得以应用的重要突破，在不同的领域中对 ADP 这项技术有不同的叫法：

1. reinforcement learning：人工智能，或者机器学习领域的研究者们叫它强化学习，这个领域的研究者们关注的是让计算机从数据中学习如何像人类一样进行决策，他们是从机器学习的角度来思考问题并进行研究的
2. neuro-dynamic programming：运筹学与优化领域的研究者叫它神经动态规划，从数学的角度上来进行研究 (笔者注：实际上很多人叫这个领域的近似动态规划就叫近似动态规划 (approximate dynamic programming, ADP)，教授叫它神经动态规划的原因是教授创立的体系中，它的名字叫神经动态规划，名字其实不重要，明白不同领域的研究重点才是问题的核心。)，关注的重点是如何能够找到问题的最优解或者尽可能好的解。
3. adaptive dynamic programming：控制领域的研究者们给它起的名字是自适应动态规划，这是从自适应控制 (adaptive control) 发展而来的技术，缩写也是 ADP，有时候也叫 ADP 控制，实际上如果你是一个控制领域的 ADP 研究者，介绍自己的时候说自己的研究方向是 RL 也是没有问题的，同样的原因，名字并不重要，重要的是领域之间的区别，控制领域关注得更多的是系统的收敛性与稳定性收敛指的是系统状态与控制最后能稳定 (与稳定性不是一回事) 收敛到某个值，同时这个系统的状态与控制最终稳定在 0，比较常见的就是最优控制，指标函数一般是状态和控制的二范数累加值，如果状态与控制不能稳定收敛到 0，指标函数就会无界，所以一般要求稳定收敛到 0，但是不排除其他情况可以不稳定收敛到 0，能够满足控制指标就可以了。

这个课程中我们主要关注优化问题中 n 个状态的折扣模型，这种问题很简单，但是实例中状态数量会非常大，这时候问题同样很难求解。求解 $n-state$ 折扣问题的方法可以扩展到其他动态规划问题，比如连续空间，连续时间与非折扣问题中去，当然这些问题更难以求解。

比较常见的近似方法有两种：问题近似和基于仿真的近似，

问题近似指的是需要求解的原问题非常复杂难以求解，比如状态空间非常大之类的，然后忽略掉一些原问题复杂的东西构造原问题的近似问题，这个近似问题的解也是原问题的可行解，但不一定是最优的，这个近似问题要求能够使用精确算法求解。这样就可以通过求解近似问题来获得原问题的近似解，这就叫做问题近似，指的是找到一个近似的问题然后使用精确算法进行求解。

基于仿真的近似指的是直接近似原问题的某些指标，然后使用这些指标进行求解，处理的还是原问题，使用近似算法直接获得原问题的近似解。

基于仿真的近似主要有三种方法，比较基础并且常用的也是这三种方法，包括 Rollout，值空间内的近似于策略空间内的近似，值空间内的近似是使用

参数化的方法近似成本函数 J，然后使用 J 进行求解，策略空间内的近似是使用参数方法直接近似最优策略，然后寻找这个近似策略的参数。

这三种近似方法都是基于仿真的，使用仿真的动机是原问题状态空间与控制空间非常大，精确算法失效，使用近似算法对状态与控制空间内具有代表性的状态和控制进行参数调优来估计整个状态与控制空间的情况。

下面要给出一个 DP 领域发展情况的大事记，课件中没有，但是笔者认为能够更好地让人了解这个领域，所以做了一回搬运工。

时间	事件	文章
1953	Richard Bellman 提出贝尔曼条件 (Bellman condition),这是强化学习的基础之一	Bellman, R. (1953). An introduction to the theory of dynamic programming.Rand Corporation Santa Monica Calif.
1956/1957	Bellman 介绍了动态规划和马尔可夫决策过程 (MDP) 的概念	Bellman, R. (1957). A Markovian decision process. Journal of Mathematics and Mechanics, 679-684.
1963	Andreae 开发出 STeLLA 系统 (通过与环境交互进行试错学习)；Donald Michie 描述了 MENACE(一种试错学习系统)	Andreae, J. (1963). STELLA: A Scheme for a Learning Machine.
1975	John Holland 基于选择原理阐述了自适应系统的一般理论	Holland, J. H. (1992).Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.MIT Press.
1977	Werbos 提出自适应动态规划 (ADP)	Werbos, P. (1977). Advanced forecasting methods for global crisis warning and models of intelligence. General System Yearbook, 25-38.
1988	R.S.Sutton 首次使用时间差分学习 (TD 算法诞生)	Sutton, R. (1988). Learning to predict by the methods of temporal differences.Machine Learning. 3 (1): 9–44.
1989	Watkins 提出了 Q 学习	Watkins, C. J. C. H. (1989). Learning from delayed rewards (Doctoral dissertation, King's College, Cambridge).
1991	Lovejoy 研究了部分可观测马尔可夫决策过程 (POMDP)	Lovejoy, W. S. (1991).A survey of algorithmic methods for partially observed Markov decision processes.Annals of Operations Research. 28(1):47–65.

continued on next page

continued from previous page

时间	事件	文章
1994	Tesauro et al. 将强化学习和神经网络结合到了一起	Tesauro, G. (1995).Temporal Difference Learning and TD-Gammon.Communications of the ACM. 38 (3).
1994	Rummery 提出 SARSA	Rummery, G. A., & Niranjan, M. (1994). Online Q-learning using connectionist systems (Vol. 37). University of Cambridge, Department of Engineering.
1996	Bertsekas 提出神经动态规划 (Neuro-Dynamic Programming)	Bertsekas, Dimitri P., and John N. Tsitsiklis. "Neuro-dynamic programming: an overview." Decision and Control, 1995., Proceedings of the 34th IEEE Conference on. Vol. 1. IEEE, 1995.
1999	Thrun 等人提出蒙特卡罗定位方法	Dellaert, F.; Fox, D.; Burgard, W. and Thrun, S. (1999). Monte Carlo localization for mobile robots.Proceedings 1999 IEEE International Conference on Robotics and Automation. 2: 1322-1328.
2009	F.L. Lewis 等人提出 integral reinforcement learning 框架解决 RL 难以应用在连续时间系统中的问题	Vrabie D, Pastravanu O, Abu-Khalaf M, et al. Adaptive optimal control for continuous-time linear systems based on policy iteration[J]. Automatica, 2009, 45(2): 477-484.
2013	Mnih et al. 提出深度 Q 学习 (DQN)	Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.
2014	Silver 提出确定性策略梯度学习 (Policy Gradient Learning)	Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D. et al .(2014). Deterministic Policy Gradient Algorithms. ICML.
2016	AlphaGo(Silver et al.) 成为深度强化学习应用的著名案例	Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484-489.

continued on next page

continued from previous page

时间	事件	文章
2016	Van Hasselt, H., Guez, A. 使用双 Q-learning 的深度强化学习	Van Hasselt, H., Guez, A., & Silver, D. (2016, February). Deep Reinforcement Learning with Double Q-Learning. In AAAI (Vol. 16, pp. 2094-2100).
2017	DeepMind 公司发布 AlphaZero 论文，进阶版的 AlphaZero 算法将围棋领域扩展到国际象棋、日本象棋领域，且无需人类专业知识就能击败各自领域的世界冠军	Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... & Lillicrap, T. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv preprint arXiv:1712.01815.
2017	Sutton 和 Barto 等学者提出了 Q( ) 算法	Asis, K. D. et al.Multi-step Reinforcement Learning: A Unifying Algorithm.arXiv:1703.01327.
2018	DeepMind 在 Nature Neuroscience 发表新论文提出了一种新型的元强化学习算法	Wang, J. X. et al. (2018). Prefrontal cortex as a meta-reinforcement learning system. Nature Neuroscience.

在这个地方关于 F.L. Lewis 及其团队的贡献进行一下补充说明，列表中只给出了 IRL 框架的提出，解决了 RL 难以处理连续时间问题的困难，实际上还有另一个很重要的贡献，机器学习领域的研究者们不关心，或者说不是那么关心系统的稳定性，只要学习表现够好就可以了，但是控制系统要求系统一定要稳定，F.L. Lewis 和他的团队对于保证系统稳定性与使用 RL 进行控制的时候系统的稳定性分析做出了很重要的贡献，不过稳定性分析的工作的代表性文章没有找到，列表中就没有给。

4.7 WHY DO WE USE SIMULATION?

这里要详细解释一下为什么要用仿真技术，使用仿真技术主要动机有两个，第一个是涉及大量计算的问题很难求解，仿真可以使用有限的计算资源计算原本计算量非常大的问题的近似结果，第二个原因是有些问题难以写出数学模型或者写不出数学模型，这时候就可以使用仿真技术产生数据代替数学模型来计算求解问题过程中的数值。

我们现在需要完成一个计算量非常大的计算任务，比如向量内积计算，如果向量维度特别大，那么需要的计算量也非常大，或者是计算一个非常复杂的定积分的数值，数学方法难以获得积分的解析形式，这时候也可以使用仿真技术 (比较常用的是蒙特卡洛方法) 求这个定积分。

假设我们想要在非常大的状态空间 (一共 n 个状态) 内计算他们的总和

或者期望值，这时需要从第一项加到第 n 项，如果 n 特别大的时候，需要的时间非常长，这时候就可以使用仿真来计算这个累加的结果。考虑一个累加计算 $\sum_{i=1}^{n} a_i$，n 非常大的时候难以计算，这时候可以根据一个分布 $\xi = \{\xi_0, \xi_1, \cdots, \xi_n\}$ 对 a_i 进行采样，然后根据这些样本计算采样的期望，数学表达推导如下：$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} \xi_i \frac{a_i}{\xi_i} = E_\xi \left\{ \frac{a_i}{\xi_i} \right\}$，也就是说，$a_i$ 的累加结果数学上严格等于 $\frac{a_i}{\xi_i}$ 在分布 $\xi = \{\xi_0, \xi_1, \cdots, \xi_n\}$ 上的期望，计算期望也就是计算均值，可以根据分布 ξ 对 a_i 采样 k 次，计算 $\frac{a_i}{\xi_i}$ 的值然后累加，采样结束后将这个累加结果除以 k 就得到了期望值，也就是想要求的 a_i 的累加的近似结果，采样数量 k 越大，得到的结果近似程度更高。假定我们能接受的采样次数是 10^5，但状态空间里有状态 10^{10} 个状态，就可以在有限的计算资源支持下计算近似结果，这就是仿真的主要优势。需要说明的一点是，算法或者要分析的模型是未知的，但是模拟器和计算机知道这个模型，这就是假设不知道问题模型的时候，算法如何求解问题。

4.8 APPROXIMATION IN VALUE SPACE

近似动态规划的第一个近似类型是值空间内的近似，也就是使用函数 $\bar{J}(i;r)$ 来近似成本函数 J^* 或者 J_μ，然后再不同类型的算法中用 $\bar{J}(i;r)$ 代替 J^* 或者 J_μ 进行计算。

函数 $\bar{J}(i;r)$ 中的 r 是值空间近似中的参数，需要决策的也从 J^* 或者 J_μ 变成了 r，不同的 r 会导致近似函数有不同的形状，我们需要做的就是通过调整 r 的值来改变 $\bar{J}(i;r)$ 的形状让 $\bar{J}(i;r)$ 尽可能地逼近 J^* 或者 J_μ 的值。

近似函数有两个比较重要的问题，第一个问题是选择参数的形式，也叫做近似结构，可以是线性的，也可以是二次，多项式的或者更复杂的形式，这个结构的选择很重要，因为它会影响到参数的搜索空间，不同的参数空间搜索难度不同，近似能力也不同，近似结构很重要，需要很小心地选择。第二个事情是选择了近似结构之后，需要用一个比较合适的方法来调整这个参数，比如训练一个神经网络，神经网络也是一种近似结构，通过调整神经网络的参数让神经网络更好地逼近 bellman 方程的成本函数，不同的搜索方法搜索效率不同，同样会影响到近似函数的近似效果。

所以这两个问题解决得是否成功，取决于这两个问题选择得好不好，有些时候这依赖于你对问题的理解。

如果你知道这个问题的模型，就可以根据这个模型来进行近似函数参数的搜索，如果不知道这个问题的部分模型或者所有的模型都不知道，就可以使用仿真产生数据，根据数据调整近似结构的参数，实际上除了仿真技术，还有其他方法可以在不知道问题模型的时候调整参数，但是这个课程中主要关注仿真技术。

近似函数除了近似成本函数，比如 Q 函数或者 V 函数，有些时候还可以通过近似成本函数 (Q 函数或者 V 函数) 的差分求解问题

4.9 APPROXIMATION ARCHITECTURES

近似结构被分为线性结构和非线性结构，这依赖于 r 的设计，线性结构由于具有凸性，因此很容易收敛并且找到最优解，但是逼近效果相比于非线性结构较差，非线性结构一般是非凸结构，因此收敛难度很大，而且很难找到全局最优解，不过非线性结构的逼近效果很好，不过线性结构具有很强大的功能。线性结构比较好训练，有很多算法可以使用而且与非线性结构相比表现更好，不过非线性结构在实际应用的时候更好用，比如神经网络。

这里用一个计算机象棋的例子说明线性结构很强大的原因，象棋的近似结构被设计成每次观察的时候的棋盘信息当成一个状态，把所有能走的位置作为决策，这时候就有了一个 MDP 模型，接下来就可以计算每个决策的得分 (比如 Q 值) 来评估这个决策有多好了。

当前国际象棋的近似设计方法是给状态提取设计一个映射，首先预测几步，就是搜索几步，搜索结束的时候观察棋盘位置并且根据已知的信息提取特征，如果我可以把棋子放到比我的对手更多的位置上去，那么对于我来说这就是一个好的特征，比如能保证王 (类似于中国象棋中的将和帅) 的安全的特征，实际设计时，可能有 30，40 或者 50 个特征，这些特征都是用不同的权重加权计算获得的，通过对位置进行估值计算出得分，也就是起始位置和当前位置与决策的 Q 值，计算出 Q 值后，选择 Q 值最高的那个决策下子。一般来说，特征数量比较少，未知信息也比较少，如果特征数量是 30、40 或者 50，那么参数特征差不多是 30-50 个，棋盘位置的空间是一个天文数字，相比于整个国际象棋的搜索空间，需要搜索的空间变得非常小，而且国际象棋程序工作得很好。

现在国际象棋的特征设置是根据 50 年来的经验完成的，人们观察国际象棋的表现调整特征的权重，然后通过试错与调整的方式寻找更好的估值方式。在我们的研究中，估值会用一种更系统更通用的方法完成。

4.10 LINEAR APPROXIMATION ARCHITECTURES

很多人在设计近似结构的时候，提取特征使用与成本函数结构相似的方式，一般是非线性的提取规则，如果这个规则比较准确，就可以用很简单的近似结构得到很精确的逼近效果 (比较极端的情况下甚至几乎等于精确的成本函数值)，假设我们已经设计了一个比较好的特征提取规则，就可以得到特征矩阵 Φ，然后就可以使用线性结构进行近似了，近似结构被分为三部分：特征矩阵 $\Phi = \Re^{n\times s}$(n 为系统状态的数量，s 为每个状态对应的特征数量)、系统状态 i 与近似函数的梯度 r，其中系统状态 $i = 1, 2, \cdots, n$，函数 $\phi(i) = \Re^s$ 表示特征 i 对应的特征列向量，$\phi(i)'$ 表示特征行向量，这样就可以使用 $\tilde{J}(i; r) = \phi(i)'r, i = 1, 2, \cdots, n$ 近似状态 i 的成本，另一种一般性的表示

方法是：

$$\tilde{J}(r) = \Phi r = \sum_{j=1}^{s} \Phi_j r_j$$

有些材料中，$\Phi(i)$ 也被叫做基函数。这种方法其实是在另一个更小的空间 $r \in \Re^s$ 中搜索成本函数，近似结构还有很多其他例子，比如仿射函数近似和径向基函数近似等。

4.11 ILLUSTRATIONS: POLYNOMIAL TYPE

下面用多项式近似和插值两个例子来解释一下线性近似结构是什么样的。

多项式近似：如果要用二次近似函数来近似成本函数，有 q 维的系统状态 $i = (i_1, i_2, \cdots, i_q)$，定义常系数项、线性项与二次项特征函数 (也叫基函数)$\phi_0(i) = 1, \phi_k(i) = i_k, \phi_{km}(i) = i_k i_m, k, m = 1, 2, \cdots, q$，就有近似函数

$$\tilde{J}(i; r) = r_0 + \sum_{k=1}^{q} r_k i_k + \sum_{k=1}^{q} \sum_{m=1}^{q} r_{km} i_k i_m$$

近似参数 r 有三种：常数项参数 r_0、一次项参数 r_k 与二次项参数 r_{km}，一共有 $1 + q + q^2$ 个参数需要优化，也就是说这时候搜索空间为 $\Re^{\left(1+q+q^2\right)}$，而原问题在系统状态 i 的每个维度都有 p 个值可以选择的时候，搜索空间为 $\Re^{p^q}$，即成本函数数量与系统状态数量相同，有 p^q 个，每个成本函数的取值范围为实数 $\Re$，需要搜索的解是一个 p^q 的实向量。一般情况下，p 会非常大，这种设计的搜索空间就远小于原问题的搜索空间，如果 p 和 q 都很小，不需要使用近似手段，精确算法就可以求解了，如果出现了 p 很小的情况，近似搜索空间还是远小于原问题的搜索空间，这里就不给出证明了，自行想象一下这几种情况就可以了。

另一个方法是**插值法 (interpolation)**，从状态空间中选择特殊并且具有代表性的状态子集 I，然后使用近似参数 $r_i, i \in I$ 来近似状态 i 对应的成本函数 $\tilde{J}(i; r) = r_i, i \in I$，对于不在状态子集 I 中的状态 i，使用插值法 (片段插值、线性插值、多项式插值、样条曲线插值、三角内插法、有理内插、小波内插等方法) 进行估计，然后只需要调整近似参数 r 就可以了。原问题的搜索空间这里就不给了，可以参见多项式近似的原搜索空间，这种方法减小搜索空间的手段是控制状态子集合的大小来控制近似参数的维度，从而减小搜索空间。

4.12 A DOMAIN SPECIFIC EXAMPLE

举一个俄罗斯方块的例子，俄罗斯方块的游戏规则这里就不详细介绍了，简单地说，不同形状的物体从顶上向下移动，你可以控制它旋转或者移动落在

墙上，组成一条横线的时候这条横线消失同时得到一分，游戏的目标是尽可能长时间地玩下去并获得尽可能多的积分，当墙的高度等于游戏界面的高度的时候，游戏就结束。这个游戏可以按照动态规划问题处理，初始状态是一个空板，板子上有很多小格子，每个格子都是 0(表示格子是空的，如果格子是满的，就用 1 表示)，整个面板有 200 个格子，这样所有的状态数量就是 2^{200}，比整个宇宙中的原子数量还要多。

一个随机形状的物体从顶部掉下来，有很多操作可以执行，平移或者旋转，大概有 30 个决策可以选择，这是一个状态空间特别大，决策空间比较小的随机问题，这个问题的目标就是最大化期望得分，定义从状态 i 开始的最大期望得分 $J^*(i)$，则 $J^*(0)$ 表示从初始状态，也就是空面板开始的最优得分。实际上，没有人知道当前状态下怎么玩是最优的，同样，也没有人或者计算机能算出 $J^*(i)$ 到底是多少。现在我们能做的就是使用一种基于仿真的算法使用 $\tilde{J}(i;r)$ 来估计 $J^*(i)$ 的值来选择决策，这个问题已经被 MIT 的一些人研究超过 20 年了。

这个问题的规模非常大，但是 MIT 的研究人员发现只需要 22 个特征就可以比较好地描述这个问题，然后使用策略迭代算法训练一个游戏策略，也就是从任意一个策略开始，使用这 22 个特征来对 $J^*(i)$ 进行近似，迭代调整相应的参数，实验中平均得分从 30 开始，逐渐到 3000，4000 和 5000，这是最近这 15 年最好的结果了。

后来开始使用其他方法，不使用策略迭代的方法得到了 70 万与 90 万的得分，尽管 MIT 的研究人员得到了很好的结果，差不多是这 22 个特征能得到的最高的分数了，但是他们仍然不知道最大期望得分是多少，这个得分几乎不可能知道。

这 22 个特征是这样的，假设有 10 列方块，这 10 列方块的高度就会产生 10 个特征，这些列的高度差能够提供 9 个特征，已有的墙中的空洞能够产生一些特征，教授说他忘记这个特征的数量是多少了，反正最后一共有 22 个特征 (如果只有这三部分的话，那空洞的特征应该有 3 个，)，这样即使系统状态空间非常大，这 22 个特征也可以在低维空间来进行计算，还能得到很好的效果，这就是一个标准的近似结构，其他例子还有国际象棋什么的，就不展开了。

4.13 APPROX. PI - OPTION TO APPROX. J_μ OR Q_μ

之前讲的是对成本函数进行近似，也就是对值迭代算法的近似是怎么用的，这一页要说一下值空间的策略迭代是怎么工作的，首先是根据当前策略 μ 对状态成本函数 $J_\mu(i;r)$ 进行近似，计算近似函数值 $\tilde{J}(i;r)$ 直到停止，然后使用 $\tilde{J}(i;r)$ 进行策略改进产生新的策略 $\bar{\mu}$，然后继续进行策略评价-策略改进的循环，直到策略迭代算法收敛。

对于策略 μ 下 Q 值的近似与上面对 J 值的近似相似，使用值迭代算法计算策略 μ 对应的 $\tilde{Q}_\mu(i,u;r)$，然后使用策略改进产生新的策略 $\bar{\mu}(i)$，同样进行策略评价-策略改进的循环，直到策略迭代算法收敛。

之前在精确动态规划的策略迭代算法中已经给出过证明，对于任意一个初

始策略 μ，都会收敛于相同的最优策略，当然近似值做策略迭代不能保证收敛到最优策略，只能说可以收敛到某一个策略。

4.14 APPROXIMATING J^* OR Q^*

J^* 与 Q^* 的关系是：

$$Q^*(i,u) = g(i,u) + \alpha \sum_{j=1}^{n} p_{ij}(u) J^*(j)$$
$$J^*(i) = \min_{u \in U(i)} Q^*(i,u)$$

如果知道了 J^* 或者 Q^* 中的任意一个，就可以根据这个关系计算另一个，另一种计算成本函数 $(\tilde{J}(i;r))$ 的方法是根据 bellman 误差，最小化表达式 $\min_r E_i \left\{ \left(\tilde{J}(i;r) - (T\tilde{J})(i;r) \right)^2 \right\}$ 调整参数 r 来让逼近效果最好，需要说明的是，这个期望计算的是系统状态满足某种概率分布时的期望。

第三种计算近似成本函数的方法是 Approximate Linear Programming(ALP)，比较有代表性的学者包括张丹、等这里不展开讨论。

与精确最优值 J 和 Q 的关系相同，近似的 $\tilde{J}$ 和 $\tilde{Q}$ 具有同样的关系，近似 J 的方法同样可以用来近似 Q 的值，这个近似值的方法可以用于值迭代，也可以用于策略迭代的策略评价过程。

4.15 APPROXIMATION IN POLICY SPACE

本页要对策略空间中的近似做一下简单的介绍，基本思路是将策略参数化，使用一参数 $r = (r_1, r_2, \cdots, r_s)$ 参数化函数 $\mu(i;r)$ 来计算控制 u，比如多项式策略函数 $\mu(i;r) = r_1 + r_2 \cdot i + r_3 \cdot i^2$ 和基于特征的线性策略 $\mu(i;r) = \phi_1(i) \cdot r_1 + \phi_2(i) \cdot r_2$。这时候需要做的就是搜索成本最小的参数 r，每一组参数 r 都对应一个平稳策略，计算出近似函数 $\tilde{J}(i;r)$ 的值之后，根据状态 i 的出现概率分布 $P = (p_1, p_2, \cdots, p_n)$ 计算期望成本 $\sum_{i=1}^{n} p_i \tilde{J}(i;r)$ 此时目标还是通过调节参数向量 r 最小化期望成本 $\sum_{i=1}^{n} p_i \tilde{J}(i;r)$，这时候搜索空间从成本函数空间变成了参数空间，搜索空间会变小，搜索方法很多，随机搜索，梯度的方法 (梯度下降共轭梯度什么的)，还有其他连续空间内搜索的方法。

对于计算成本函数 J 来说，参数化策略是一种间接方法，通过对策略进行搜索来寻找最优的成本函数，需要先搜索策略，再根据策略计算相应的策略成本，但是对于决策来说，我们的目的实际上是找到合适的控制，而先计算成

本函数再计算控制变成了间接方法，参数化策略不需要计算成本函数，可以直接输出控制。

4.16 DIRECT POLICY EVALUATION

策略 μ 的成本函数 J_μ 维度非常大，搜索空间也非常大，根本没法算出精确解，所以要把原问题的 J_μ 空间使用算子 Π 降维映射到低维度空间 ΠJ_μ 进行搜索，其实就是使用近似结构在参数空间中搜索近似效果最好的近似函数。课件中的图想表达在原成本函数空间的子空间 $S = \{\Phi r | r \in \Re^s\}$ 中进行搜索。

对于任意的特征函数 Φ，参数 r 在空间中 $r \in \Re^s$ 对应的 ΠJ_μ 集合理论上与原问题的 J_μ 集合是相等的集合，但是对于给定的特征函数 Φ，参数 r 在空间中 $r \in \Re^s$ 对应的 ΠJ_μ 集合就不能保证与原问题的集合 J_μ 相等了，现在需要做的有两件事，第一件事是给一个合适的特征函数 Φ，然后需要在这个 Φ 的基础上在参数空间中搜索最优的参数 r 让近似效果最好，可以理解成需要解决一个回归问题，使用最小二乘模型求解参数 r，最小二乘模型需要知道各近似成本函数的值，这个值可以通过各种方法求解，比如仿真技术之类的方法估计成本函数，估计成本函数后使用这些值计算近似效果最好的参数 r，然后使用这个参数 r 计算其他状态下的成本，如果近似结构是非线性结构，也可以使用同样的思路和方法进行近似。

这地方需要详细介绍一下算子 Π，因为笔者在不同的时间段看了这部分三四次都没有看明白，后来找到了一篇文献[6]，按照笔者的理解来介绍一下这个概念。在参考文献中，Π 是一个依赖于概率分布 ξ 的矩阵，没有说 Π 的形状，但是根据公式表示，应该是一个 $\Re^{n\times n}$ 的矩阵，其中 n 是系统状态的数量。教授说的映射到低维度空间里，实际上是想说从原始空间，即成本函数向量空间中搜索，变成在低维度参数空间中搜索，然后使用映射算子反向恢复成原问题的成本函数矩阵，这样从整个问题的角度来说，就变成了高维度空间到低维度空间的映射，也就是从原高维空间 J_μ 使用算子 Π 到新的低维空间 $r \in \Re^s$ 的含义。当然这个降维映射会造成精度损失，也是用精度换可求解性的方法了。

4.17 DIRECT EVALUATION BY SIMULATION

在使用投影 (其实可以理解成近似) 求 bellman 方程次优解的时候，如果系统模型不可知或者模型规模庞大，可以使用仿真方法估计成本函数的值，由于仿真采样无法获取所有状态的样本，所以只能使用不完全采样的结果进行估算，同时采样使用的概率分布可能不准确，所以在估值的时候需要使用加权欧几里得范数 (weighted Euclidean norm)$||\cdot||_\xi$，这个加权欧几里得范数 (不知道数学上叫什么，这个叫法是直接从课件翻译来的) 的定义是 $||x||_\xi = \sum_{i=1}^{n} \xi_i \left(x_i\right)^2$

也就是说，最优参数 r^* 可以通过表达式得到

$$\arg\min_{r\in\Re^s} ||\Phi r - J_\mu||_\xi^2 = \arg\min_{r\in\Re^s} \sum_{i=1}^{n} \xi_i \left(\phi(i)'r - J_\mu(i)\right)$$

这个最小化问题是一个最小二乘问题，最小二乘问题从理论上给了很好的结果，可以直接从梯度等于 0 得到最优解

$$r^* = \left(\sum_{i=1}^{n} \xi_i \phi(i)\phi(i)'\right)^{-1} \sum_{i=1}^{n} \xi_i \phi(i) J_\mu(i)$$

即使经过近似后，最优参数 r^* 由于状态空间巨大依然难以计算，同时这个矩阵求逆计算是逆矩阵的期望，之前讲过的使用仿真技术代替大规模计算就可以使用了，使用仿真技术使用状态概率分布 ξ 采样 k 次，得到 k 个样本 $\{(i_1, J_\mu(i_1)), (i_2, J_\mu(i_2)), \cdots, (i_k, J_\mu(i_k))\}$，然后使用蒙特卡洛方法计算 r 的期望值 $\hat{r}$ 来代替最小二乘的最优参数 r^*，直接对策略成本投影，很好用但不是最好的方法。

4.18 INDIRECT POLICY EVALUATION

这一页的介绍中教授说可以使用 Galerkin approximation 进行近似，没能理解 Galerkin approximation 是什么，只能通过个人的理解说说这一页是什么思路。

在直接策略评价近似中，直接使用近似函数近似通过蒙特卡洛估计出的 J_μ，而间接近似中，还是使用近似函数近似成本函数，但是这时候近似的目标值就不是蒙特卡洛估计出的 J_μ 了，还是近似函数近似的目标值，也就是说求解的是用近似函数代替 J_μ 产生的 bellman 方程，求解这个近似 bellman 方程的方法有很多，比较主要的有 Temporal Difference(λ)(TD(λ))[11]、Least Squares Temporal Difference(λ)[8] [9], (LSTD(λ)) 和 Least Squares Policy Evaluation(λ)(LSPE(λ))[10] 等。

这部分不是很明白，先放着，需要再查点东西才能继续补充。

4.19 BELLMAN EQUATION ERROR METHODS

使用 bellman 误差法求解近似梯度，需要最小化该表达式 $\min_t ||\Phi r - T_\mu(\Phi r)||_\xi^2$，教授说这也是一种特殊的 Galerkin approach，这个方法工作的时候流程是这样的：根据分布 ξ 产生状态 i_k，根据策略 μ 产生状态转移样本 (i_k, j_k)，构造上面的最小化表达式或者投影问题的最优条件，使用蒙特卡洛方法求解最优条件或者投影方程。

工作流程有两个问题，第一个是产生样本，如果使用固定的分布，很容易出现的情况是状态集中出现在某个区域内，这样蒙特卡洛平均得到的成本就不准确，需要尽可能随机地产生状态，第二个问题是求解参数的时候，大规模计算非常消耗计算资源，这时候可以使用蒙特卡洛平均代替大规模线性计算。

4.20 ANOTHER INDIRECT METHOD: AGGREGATION

一个系统有离散状态 $1,2,\cdots,n$，选择相似的状态用一个新状态 x 表示，给定这个新状态 x 的成本函数 r，就会产生新状态 $x_1,x_2,\cdots,x_s$ 和新的成本函数 $r=(r_1,r_2,\cdots,r_s)$，然后计算这些新状态的成本函数值，也就是构造了一个原问题的近似问题，比如课件上的那个图，把状态 $1,2,4,5$ 聚合成状态 x_1，状态 $3,6$ 聚合成状态 x_2，状态 $7,8$ 和 9 分别聚合成 x_3 和 x_4，原动态规划模型状态与成本从 $s=(1,2,3,4,5,6,7,8,9)$ 与 $J=(J(1),J(2),J(3),J(4),J(5),J(6),J(7),J(8),J(9))$ 变成新的动态规划模型的状态与成本函数 $x=(x_1,x_2,x_3,x_4),r=(r_1,r_2,r_3,r_4)$，这个例子中，聚合列分布矩阵 Φ 与聚合行分布矩阵 D

$$\Phi=\begin{bmatrix}1&0&0&0\\1&0&0&0\\0&1&0&0\\1&0&0&0\\1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\quad D=\begin{bmatrix}1/4&1/4&0&1/4&1/4&0&0&0&0\\0&0&1/2&0&0&1/2&0&0&0\\0&0&0&0&0&0&1/2&1/2&0\\0&0&0&0&0&0&0&0&1\end{bmatrix}$$

这里要多解释一下，课件解释的不是很清楚，需要参考一下辅助材料[4]，假定有聚合状态 q 个，原始状态 n 个，聚合矩阵 Φ 表示原始状态 i 与聚合状态 ℓ 之间的对应关系，如果原始状态 i 属于聚合状态 ℓ，则有 $\phi_{i\ell}=1$，否则值为 0，也就是说聚合矩阵是一个 $\{0,1\}^{n\times q}$ 的矩阵。反聚合矩阵 D 有 q 行 n 列，具体描述为观察到聚合状态 ℓ 时，实际状态为 i 的概率 $d_{\ell i}$，聚合状态需要满足 $\sum_{\ell=0}^{q}d_{\ell i}=1$，即某个聚合状态对应所有原始状态的概率之和为 1。所以就有了表达式 $\Phi r=\Phi DT_\mu(\Phi r)$，从这个表达式出发的话，也可以把 (ΦD) 当成一个投影来处理，状态聚合规则一旦设计好，聚合矩阵 Φ 就已经确定了，所以个人理解，其实可以写成 $r=DT_\mu(\Phi r)$，有些方法里，D 和 r 都是需要调整的，某次迭代时算出了当前的 D 和 r，计算 Φr 可以得到原系统状态对应的成本，然后对这个原始成本使用算子 T_μ，得到新的成本，在乘一个反聚合矩阵 D，得到 r 的期望值，再继续更新。

教授说的 D 减少行的数量，Φ 减少列的数量，没能完全理解，这个近似的本质是减少系统状态的数量，一旦系统状态数量减少，bellman 方程组的行和

列都会相应减少，因为 bellman 方程组的系数矩阵是一个方阵，行和列的数量相等，是系统状态的数量。

4.21 AGGREGATION AS PROBLEM APPROXIMATION

这一页就没有什么新的内容了，主要内容都在上一页给出了，聚合其实是做问题近似，从高维度的问题变成低维度的问题，然后使用精确方法求解这个低维度的问题，再使用 Φ 得到原问题的近似成本函数进行控制，从原问题到聚合问题的转换图已经解释得很直观了，这里就不展开了，需要说明的是，有时候聚合问题是可以使用精确方法求解的，如果聚合问题无法用精确方法求解 (聚合问题维度依然很高或者模型不可知之类的)，需要使用仿真方法估算聚合问题的成本，需要注意的就是需要使用原系统的仿真进行采样，从原系统采样，转换成聚合状态，估算，再转回原问题，继续采样。

聚合问题有一个限制，就是近似结构收到限制，因为聚合方法需要有概率矩阵 D，这是一种比较特殊的投影方法，所以近似结构收到限制，而一般的投影方法可以使用更一般的近似结构。

4.22 THEORETICAL BASIS OF APPROXIMATE PI

在近似策略评价中，使用 δ 进行近似策略评价，使用 ϵ 进行近似策略改进，这样得到的近似成本会有一个误差上界 $\frac{\epsilon+2\alpha\delta}{(1-\alpha)^2}$，算法得到的策略的成本函数也会在这个区间内，通常实现算法的时候可以从任意一个策略开始，不断改进，最后进入靠近最优解附近的趋于，并且有可能产生震荡现象，投影法的状态转换容易导致震荡，而聚合法完全不会震荡，原本会震荡的状态最终会到达同一个聚合状态中。

4.23 THE ISSUE OF EXPLORATION

评估策略 μ 的成本时，需要使用这个策略产生样本，而这样采样会导致状态比较集中地出现，不经常出现的状态估值会产生很大的误差，这就是不充分探索导致的误差，确定性系统这种现象尤其严重，所以充分采样就比较重要。

有几种方法可以避免不充分采样：1. 有规律地使用有代表性的初始状态重新开始仿真轨迹; 2. 随机选择控制进行不确定的状态转移; 3. 使用两个马尔科夫链进行采样，一个用来产生状态，另一个用来进行状态转移其实这个方法已经有点 off-policy 的思想了，或者说这种方法就是 off-policy 的方法。

4.24 APPROXIMATING Q-FACTORS

之前讨论过 Q 函数，这里就不过多重复了，与 J 函数相似，不充分探索同样会造成很严重的问题，使用 Q 函数求解时，最需要考虑的就是探索方法。

4.25 STOCHASTIC ALGORITHMS: GENERALITIES

想要求解一个线性方程组 $x = Ax + b$，如果系数矩阵不可知，就没有办法通过 $x = (I - A)^{-1} + b$ 求解，只能通过采样的方式估计 A 与 b 的值，假定观测到 m 个受到仿真噪声 w_k 和 W_k 干扰的样本 $b + w_k$ 和 $A + W_k$，有很多种方法可以估计 A 和 b 的值或者直接算出解 x 的值。这里给出两种方法：随机近似与蒙特卡洛估计。

1. Stochastic Approximate (SA) Approach: 采样时对于样本 $k = 1, 2, \cdots, m$，使用

$$x_{k+1} = (1 - \gamma_k)x_k + \gamma_k \left((b + w_k) + (A + W_k)\, x_k\right)$$

   计算 x 的值，这种方法可以一边采样一边计算，下面的蒙特卡洛估计就只能采样之后把获得的样本一起放进算法中

2. Monte Carlo estimation (MCE) approach: 获取 m 个样本后，分别使用 $b_m = \frac{1}{m} \sum_{k=1}^{m} (b + w_k)$ 与 $A_m = \frac{1}{m} \sum_{k=1}^{m} (A + W_k)$ 估计 b 与 A 的值，然后用这两个值计算相应的 $x_m = (1 - A_m)^{-1} b_m$

SA 与 MCE 还有很多其他算法，比如 SA 的两个实现 TD(λ) 和 Q-learning 与 MCE 的两个实现 LSTD(λ) 和 LSPE(λ)。

这种求解线性方程组的方法，可以用来对动态规划算法中的策略评价或值迭代过程中的线性方程组进行求解。

4.26 COSTS OR COST DIFFERENCES?

同一个系统处于状态 x 的时候执行两个不同的控制 u 和 u' 分别有期望成本 $E\{g(x, u, w) + \alpha J_\mu(\bar{x})\}$ 与 $E\{g(x, u', w) + \alpha J_\mu(\bar{x}')\}$ 其中有 $\bar{x} = f(x, u, w)$ 与 $\bar{x}' = f(x, u', w)$，在选择状态 x 下的控制时，依据的是两个控制导致的成本值，也就是哪一个控制带来的成本更小，把这两个成本相减可以得到 $E\{(g(x, u, w) - g(x, u', w)) + \alpha\,(J_\mu(\bar{x}) - J_\mu(\bar{x}'))\}$，定义新符号 D_μ 与 $G_\mu(x, x', w)$：

$$D_\mu(\bar{x},\bar{x}') = J_\mu(\bar{x}) - J_\mu(\bar{x}')$$
$$G_\mu(x,x',w) = g(x,\mu(x),w) - g(x',\mu(x'),w)$$

如果直接使用状态 x 下的各控制的 Q 值做比较选择最小的 Q 值对应的控制，理论上可以最小化期望成本，但是这个时候得到的近似成本是带有噪声的 (该系统是随机系统，如果是确定性系统，就没有噪声了)，用两个带有噪声的成本值作比较或者相减，并不能得到太准确的相对大小关系，所以这时候引进符号 D 表示两个状态成本的差值，这个时候就不受到噪声的影响了。

不加解释地给出表达式

$$D_\mu(x,x') = E\left\{G_\mu(x,x',w) + \alpha D_\mu(\bar{x},\bar{x}')\right\}$$

这个表达式是满足 bellman 的各种性质的，这里就不给出解释了，把 D 和 G 的定义带入就可以看到满足 bellman 各种性质了。想要求 D 的值，之前讲的所有方法都可以使用，包括精确算法与近似算法，这也是误差学习的思路。

4.27 AN EXAMPLE (FROM THE NDP TEXT)

这一页给了一个例子来解释学习成本的梯度或者差值效果比直接学习成本值效果更好，假设有一个系统，状态方程 $x_{k+1} = x_k + \delta u_k$，即时成本 $g(x,u) = \delta(x^2+u^2)$，其中 $\delta > 0$ 且 $\delta \to 0$。其实这个系统是从一个连续时间系统离散化得到的，原始的连续系统模型为

$$J(x(t)) = \int_t^\infty (x(\tau)^2 + u(\tau)^2)d\tau$$
$$\dot{x(t)} = x(t) + u(t)$$

这里还是按照课件给的，使用离散系统的方法求解，离散化之后的离散系统为

$$J(x(k)) = \sum_{t=k}^{\infty} \delta(x(t)^2 + u(t)^2)$$
$$x(k+1) = x(k) + \delta(u(k))$$

使用控制策略 $\mu(x) = -2x$ 进行控制，可以得到状态与控制关于初始状态 $x_0 = x$ 的表达式 $x_k = (1-2\delta)^k x$，$u_k = -2(1-2\delta)^k x$，将 u_k 与 x_k 代入上式的 $J(x)$ 可得

$$\begin{aligned}J(x)=&\sum_{t=0}^{\infty}\delta(x_t^2+u_t^2)=\sum_{t=0}^{\infty}\delta(x_t^2+(-2x_t)^2)=\sum_{t=0}^{\infty}5\delta(x_t^2)\\=&\sum_{t=0}^{\infty}5\delta((1-2\delta)^k x)^2=5\delta(x)^2\sum_{t=0}^{\infty}(1-2\delta)^{2k}=5\delta x^2\sum_{t=0}^{\infty}(1-2\delta)^{2k}\end{aligned}$$

记 $S=\sum\limits_{t=0}^{\infty}(1-2\delta)^{2t}$ 可知这是一个首项 $a_1=1$，等比通项 $q=(1-2\delta)^2$，末项 $a_n=(1-2\delta)^{2n}$ 的等比数列，由等比数列求和公式可知

$$S=\frac{a_1(1-q^n)}{1-q}=\frac{1-q^n}{1-q}=\frac{1-(1-2\delta)^{2n}}{1-(1-2\delta)^2}$$

代回指标函数可得

$$J(x)=5\delta x^2\frac{1-(1-2\delta)^{2n}}{1-(1-2\delta)^2}=5\delta x^2\frac{1-(1-2\delta)^{2n}}{4\delta-4\delta^2}=\frac{5x^2}{4}\frac{1-(1-2\delta)^{2n}}{1-\delta}$$

由于该问题为无限期问题，所以时间趋于无穷大，则有

$$J(x)=\lim_{n\to\infty}\frac{5x^2}{4}\frac{1-(1-2\delta)^{2n}}{1-\delta}=\frac{5x^2}{4}\frac{1}{1-\delta}$$

将 $\frac{1}{1-\delta}$ 在 $\delta=0$ 处做泰勒展开可以得到 $\frac{1}{1-\delta}=1+\delta+\delta^2+\cdots$，代入上式可得 $J(x)=\frac{5x^2}{4}(1+\delta+\delta^2+\cdots)=\frac{5x^2}{4}(1+\delta)+\mathcal{O}(\delta^2)$，使用 $J(x)$ 可以得到

$$\begin{aligned}Q(x_t,u_t)=&g(x_t,u_t)+J(x_{t+1})=\delta(x_t^2+u_t^2)+\frac{5(x_t+\delta u_t)^2}{4}(1+\delta)\\=&\frac{5}{4}x^2+\delta(\frac{9}{4}x^2+u^2+\frac{5}{2}xu)+\delta^2(\frac{5}{4}u^2+\frac{5}{4}\delta u^2+\frac{5}{2}xu)\\=&\frac{5}{4}x^2+\delta(\frac{9}{4}x^2+u^2+\frac{5}{2}xu)+\mathcal{O}(\delta^2)\end{aligned}$$

由于 x 已知，所以成本函数 J 会受到 u 的影响，影响到 J 的与控制相关的值是 $\delta\left(u^2+\frac{5}{2}xu\right)$，由于 δ 很小 $(\delta\to 0)$，所以如果只使用函数近似成本函数 J 的值，由于近似的不准确性，不同的 u 带来的成本函数的变化就会被 $\frac{5}{4}x^2$ 的近似误差干扰导致无法区分不同控制的好坏关系，这时候近似成本函数的差值进行控制得到的成本就会好于只近似成本函数的值进行控制得到的成本。这部分推导可以参考[1]。

顺便说一句，$Q(x,u)$ 可以表达为

$$\begin{aligned}Q(x_t,u_t)=&\frac{5}{4}x^2+\delta(\frac{9}{4}x^2+u^2+\frac{5}{2}xu)+\mathcal{O}(\delta^2)\\\approx&\frac{5}{4}x^2+\delta(\frac{9}{4}x^2+u^2+\frac{5}{2}xu)\\=&\left(\frac{5}{4}+\frac{11\delta}{16}\right)x^2+\delta\left(u+\frac{5}{4}x\right)^2\end{aligned}$$

此时进行策略改进，可以得到新的策略 $\mu(x) = -\frac{5}{4}x$，可以看得到，如果近似 Advantage，$A(x,u) = \delta\left(u + \frac{5}{4}x\right)^2$，控制方案就不会受到离散因子 δ 的影响 (一般情况下，连续时间系统按照这种方法离散化为离散时间系统会造成系统信息缺失，同样会导致成本函数与控制方案的差异)[1]。

Part IV

APPROXIMATE DYNAMIC PROGRAMMING: ISSUES

5

LECTURE 4: Approximate VI and PI

CONTENTS

5.1 LECTURE OUTLINE

之前的课程提到的精确值迭代和精确策略迭代是无限期折扣动态规划诸多算法中比较主要的两个，今天的可能会介绍在值空间内对值和策略进行近似。值空间近似指的是在更低维度的空间内对原空间内的一个函数进行近似，之前我们讨论的是使用仿真来进行近似，这里也会继续用仿真进行近似，这个近似可能是成本函数，或者问题的某个策略的成本函数。

之前的课程提到过，投影 bellman 方程，使用矩阵形式近似求解这个方程

组会产生大量的内积计算，这里要讨论如何使用仿真技术来处理这些计算需求。

本次课程的主要目录如下：

1. Review of approximation in value space
2. Approximate VI and PI
3. Projected Bellman equations
4. Matrix form of the projected equation
5. Simulation-based implementation
6. LSTD and LSPE methods
7. Optimistic versions
8. Multistep projected Bellman equations
9. Bias-variance tradeoff

5.2 DISCOUNTED MDP

有限状态集与有限控制集下的折扣 MDP 模型，已经出现过好多次了，这里就不重复了。

5.3 "SHORTHAND" THEORY –A SUMMARY

同样，速记符不重复介绍了。

5.4 THE TWO MAIN ALGORITHMS: VI AND PI

值迭代与策略迭代，本质上是对策略进行评估然后改进策略或更新成本函数，需要求解一个高维非常大规模的方程组，没办法使用高斯消元或者 matlab 求解，就需要使用其他方法来求解这个方程组，比如使用近似方法来求解。

5.5 APPROXIMATION IN VALUE SPACE

原始的动态规划中，需要求 J^* 和 J_μ，上面介绍过，规模过大的时候无法求解，就需要找到一个低维空间内的向量 $r=(r_1,r_2,\cdots,r_s)$，使用 $\tilde{J}(i;r)$ 对 J^* 和 J_μ 做近似，通过调整 r 来增强 $\tilde{J}(i;r)$ 的近似能力，这样对于原问题的维度 n 来说，r 的维度 s 就小了很多，问题的规模也减小了很多，有很多种备选的近似结构，近似算法得到的解也依赖于近似结构，所以选近似结构也是一个需要研究的问题。

在给定近似结构之后，就有了一个子问题的一步策略：

$$\tilde{\mu}(i)=\arg\min_{u\in U(i)}\sum_{j=1}^{n}p_{ij}(u)\left(g(i,u,j)+\alpha\tilde{J}(j;r)\right),\forall i$$

这个表达式实际上是切断了当前阶段和下一阶段的联系，减小了问题的规模。减小问题规模有两种理解方式，第一种是说得比较多的决策空间从控制空间变成了参数空间，搜索空间减小了所以求解变得更容易，这时候更容易求解是由于近似。另一种解释是切断了当前阶段和下面一个阶段之间的联系，这样搜索空间就只有当前控制空间了，假定有 100 期，求第一期的时候需要知道第 2 期，第 3 期直到第 100 期的成本函数值，每一期有 3 个控制可以选，这样整体的控制空间就是 3^{100}，切断之后控制空间变成了 3，这时候减小搜索靠的就是切断阶段间的联系。参数化近似的作用是直接减小了搜索空间，切断阶段间联系的作用是每次计算最优控制的时候减小搜索空间，这两个作用一起产生导致了 MDP 更容易求解，但是哪一个因素作用更大，就需要具体问题具体分析了，也许会有量化的解析表达，但是这里不给出结论，只定性分析。

大部分情况下，我们都会倾向于使用线性近似结构 $\tilde{J}(r)=\Phi r,\Phi\in\Re^{n\times s}$，得到的是成本函数向量，非线性近似结构同理，也会得到成本函数向量，使用线性结构的原因是线性结构的凸性能够保证得到全局最优解，但是同时带来的是逼近能力的降低，反过来，非线性结构无法保证得到全局最优解，不过一般情况下逼近能力比线性结构要强。

5.6 LINEAR APPROXIMATION ARCHITECTURES

对于状态 i 的近似使用公式 $\tilde{J}(i;r)=\phi' r,i=1,\cdots,n$ 计算成本值：

整个成本向量的计算公式用 $\tilde{J}(r)=\Phi r=\sum_{j=1}^{s}\Phi_j r_j$ 计算：

Φ 和 r 的定义为 $\Phi\in\Re^{n\times s}$，$r\in\Re^s$，Φ_j 表示第 j 列的向量，这个向量有 n 个元素，r_j 表示 r 的第 j 个元素，这样乘起来然后累加得到成本向量。

算法运行过程是这样的：观察系统状态 i，提取特征向量 $\phi(i)$，计算线性成本函数 $\phi(i)'r$

近似函数的结构除了线性，还有很多其他结构，比如多项式结构，径向基函数近似等。

5.7 APPROXIMATE (FITTED) VI

近似值迭代有时候也被称作拟合值迭代 (Fitted Value Iteration)，一些文献会这么叫它。值迭代是使用 $J_k(i) = (T^k J_0)(i), k = 1, 2, \cdots$ 进行迭代的（假定，上面已经谈过了，这时候搜索空间非常大，极其难以求解，这时候就使用近似成本来减小求解难度，减小求解难度的原理之前已经讨论过了，这里就不继续讨论了。

拟合值迭代算法是这样工作的：从一个初始成本向量 J_0 开始（比如 $J_0 \equiv 0$），根据 $\tilde{J}_k$ 产生 $\tilde{J}_{k+1}$(比如 $\tilde{J}_{k+1} \approx T\tilde{J}_k$) 或者是使用某个策略算子 T_μ，这时候用 $\tilde{J}_{k+1} \approx T_\mu \tilde{J}_k$ 产生新的 $\tilde{J}$，这样就不断地对 $\tilde{J}_k$ 使用算子 T，再用 $\tilde{J}_{k+1}$ 近似算子作用后的结果，经过很多次 (N 次) 迭代后，就可以得到一个比较接近成本向量的参数向量 r 了。这个算法比较老，可以追溯到五六十年代。

这个算法对于最小化算子 T 和特定策略的算子 t_μ 都适用。从 $T\tilde{J}_k$ 到 $\tilde{J}_{k+1}$ 的过程被教授叫做 projection，也就是投影，他说他通常使用欧几里得范数，下面的内容会介绍投影到底是什么。

这是个人的一点理解了，近似后要求的是参数向量 r，但是上面的流程中没有提到参数向量，个人觉得调节参数的操作应该发生在 “用 $\tilde{J}_{k+1}$ 近似算子作用后的结果” 这个过程中，对 $\tilde{J}_k$ 使用算子 T 得到的 $T\tilde{J}_k$ 会优于 $\tilde{J}_k$，这是一个数值向量，可以认为是相对接近最优解的，还有一组参数向量 r，我们希望参数向量带来的近似成本尽可能接近最优解，但是我们不知道最优解是什么，只能知道目前为止最接近最优解的成本 $T\tilde{J}_k$，所以就可以根据 $T\tilde{J}_k$ 的值和当前 r 算出来的成本向量的误差来调整 r，可以使用比如梯度下降之类的算法，但是这里面使用算法调整 r 的值是直接迭代到 r 收敛还是只迭代一步或者几步就需要研究一下了，强化学习的研究者认为直接迭代到收敛会导致只对这一批样本过拟合，对整个样本空间逼近程度不足，而自适应动态规划和以教授为首的一批研究者认为迭代到收敛会保证迭代的误差维持在某个界内。

5.8 WEIGHTED EUCLIDEAN PROJECTIONS

先给出加权欧几里得范数 $||J||_\xi$ 的定义：$||J||_\xi = \sqrt{\sum_{i=1}^{n} \xi_i (J(i))^2}$，其中参数 $\xi = (\xi_1, \xi_2, \cdots, \xi_n)$ 都是正数。定义算子 Π 是对原空间的压缩，即对 J 求加权欧几里得范数的操作。

上面分析过调整 r 的操作，一般情况下，拟合参数最常用的是最小二乘法，教授用的也是最小二乘法，表达式为：$r^* = \arg\min_{r \in \Re^s} ||\Phi r - J||_\xi^2$。这时候最小

二乘是对所有系统状态进行拟合，由于状态数量 n 过大，最小二乘可能会不工作或者工作得很慢，即使近似后也会工作得很慢，这时候就需要解决这个问题，上面说的加权欧几里得范数就是用来解决这个问题的。权重 ξ 的定义是正数，这时候就可以理解成概率分布，那么权重 ξ 就还有一个限制：$\sum \xi = 1$，这时候 ξ 就是一个关于系统状态 x 的概率分布，根据 ξ 对系统状态进行采样可以得到系统状态样本 $\{x_t\}, t = 1, 2, \cdots, k$，使用 $J(i_t)$ 和 $\phi(i_t)'r$ 进行拟合，这时候就有了新的最小二乘问题：$\min\limits_{r \in \Re^s} \sum\limits_{t=1}^{k} \left(\phi((i_t)'r) - J(i_t)\right)^2$，这时候最小二乘问题也被近似了，就可以用求解器或者随便什么方法求解了。

5.9 FITTED VI - NAIVE IMPLEMENTATION

这种方法实际上是用系统状态空间的一个子集代替整个状态空间进行拟合和逼近，使用 bellman 方程进行计算的时候，算的是期望值，如果有系统模型，就可以直接根据概率计算期望，如果没有系统模型，就可以使用仿真的方法来计算成本函数的期望，这种算法也叫做无模型 (model-free) 的方法。

如果拟合能够比较均匀准确地在拟合误差 δ 内进行，即拟合目标值和拟合函数值的误差绝对值满足：$\max\limits_{i} |\tilde{J}_{k+1}(i) - T\tilde{J}_k(i)|$，这样当迭代次数 k 趋于无穷的时候，就能够保证近似函数与最优函数之间的误差上界小于一个与 δ 相关的定值：

$$\limsup_{k \to \infty} \max_{i=1,2,\cdots,n} \left(\tilde{J}_k(i, r_k) - J^*(i)\right) \leq \frac{2\alpha\delta}{(1-\alpha)^2}$$

这种方法看起来还不错，但是下一页会给出一个失败的例子。

5.10 AN EXAMPLE OF FAILURE

有一个只有两个状态 1 和 2 的 MDP，转移概率是 1，就是说控制是从状态 1 到状态 1，最后一定会到达状态 1，转移成本恒为 0，即 $J^*(1) = J^*(2) = 0$，对于策略 $\mu(X) = (2, 2)$(即系统处于状态 1 和 2 都要向状态 2 转移)，最优参数 r 是多少。

这里使用特征矩阵 $\Phi = (1, 2)$，近似参数 $r \in \Re$，则有近似成本向量 $\tilde{J}_k = (r_k, 2r_k)$，使用权重向量 $\xi = (\xi_1, \xi_2)$ 进行投影，由上页的结果，有最小二乘问题：

$$r_{k+1} = \arg\min_{r} \left[\xi_1 \left(r - \left(T\tilde{J}_k(1)\right)\right)^2 + \xi_2 \left(2r - \left(T\tilde{J}_k(2)\right)\right)^2\right]$$

bellman 方程 $J(i) = \min_u \sum_{j=1}^{n} p_{ij}(u)(g(i,u,j) + \alpha \tilde{J}(j), \forall i$，由于策略给定，转移确定且转移成本为 0，所以这个问题的成本函数值为 $\tilde{J}_{k+1}(1) = \tilde{J}_k(2) = 2\alpha r_k$，$\tilde{J}_{k+1}(2) = \tilde{J}_k(1) = 2\alpha r_k$，带入可得：

$$
\begin{aligned}
&\xi_1 \left(r - \left(T\tilde{J}_k(1)\right)\right)^2 + \xi_2 \left(2r - \left(T\tilde{J}_k(2)\right)\right)^2 \\
=&\xi_1 \left(r - (2\alpha r_k)\right)^2 + \xi_2 \left(2r - (2\alpha r_k)\right)^2 \\
=&\xi_1 \left(r^2 - 4\alpha r r_k + 4\alpha^2 r_k^2\right) + \xi_2 \left(4r^2 - 8\alpha r r_k + 4\alpha^2 r_k^2\right) \\
=&(\xi_1 + 4\xi_2) r^2 - (4\alpha\xi_1 r_k + 8\alpha\xi_2 r_k) r + R(1) \\
=&r^2 - \frac{(4\alpha\xi_1 r_k + 8\alpha\xi_2 r_k)}{(\xi_1 + 4\xi_2)} r + \frac{(4\alpha\xi_1 r_k + 8\alpha\xi_2 r_k)}{R(1)} \\
=&\left(r - \frac{2(\alpha\xi_1 r_k + 2\alpha\xi_2 r_k)}{(\xi_1 + 4\xi_2)}\right)^2 + R'(1)
\end{aligned}
$$

其中 $R(1)$ 和 $R'(1)$ 表示一个常数项，由上式可知

$$
r = \frac{2(\alpha\xi_1 r_k + 2\alpha\xi_2 r_k)}{(\xi_1 + 4\xi_2)} = \alpha \frac{2(\xi_1 + 2\xi_2)}{(\xi_1 + 4\xi_2)} r_k
$$

令 $\beta = \frac{2(\xi_1+2\xi_2)}{(\xi_1+4\xi_2)}$，可得 $r_{k+1} = \alpha\beta r_k$。当 $\alpha > 1/\beta$(比如 $\xi_1 = \xi_2 = 1$) 的时候，r 的值会发散，这时候算法就失效了。所以设计采样权重 ξ 就很重要，ξ 的取值会直接影响到算法是否工作。

导致算法失效的原因是原 bellman 方程中算子 T 是压缩映射，而投影后 $\Pi_\xi T$ 无法保证依然是压缩映射 (实际上这不是一个压缩映射)，所以会导致有些情况下算法失效。

5.11 NORM MISMATCH PROBLEM

之前的讨论需要算子 T 是压缩映射，但是在投影拟合值迭代里面，只有 T 是压缩映射是不够的，需要 $\Pi_\xi T$ 也是压缩映射才行。

假设 T 是压缩映射，给定权重向量 ξ，如果投影和采样都使用 ξ，就不会出问题，因为这个时候 $\Pi_\xi T$ 是一个压缩映射，但是如果投影使用 ξ，采样使用另一个分布，就容易出现发散的现象，导致这种现象出现的原因是范数不匹配，也就是采样分布和投影权重不匹配。所以采样分布与投影加权系数的选择很重要，需要慎重。在近似策略迭代中也会产生相应的问题。

5.12 APPROXIMATE PI

原始的精确策略迭代有两步，策略评估和策略改进，策略评估就是对某策略执行值迭代，等同于求解一个多元一次方程组，策略改进是做 n 次最小化操作。

策略评估难以精确求解，所以需要使用近似方法求解，求解之后使用近似成本函数进行策略改进，如果使用误差上界 $\max_i |\tilde{J}_{\mu^k} - J_{\mu^k}(i)| \leq \delta, k = 0, 1, \cdots$ 进行策略评估，每一次迭代得到的策略 $\left\{\mu^k\right\}$ 带来的最大的 gap(当前成本函数与最优解的差值) 上界为：$\limsup_{k\to\infty} \max_i \left(J_{\mu^k}(i) - J^*(i)\right) \leq \frac{2\alpha\delta}{(1-\alpha)^2}$，可以看到，这样的近似策略迭代 gap 上界与近似值迭代的 gap 上界是相同的。

5.13 POLICY EVALUATION

策略评估 (值迭代) 的一种分类方式是，分成直接策略评估和间接策略评估，直接策略评估是使用仿真采样，然后使用这些样本估算成本函数。间接策略评估是使用上面谈的投影近似值迭代方法，计算投影 bellman 方程组，然后求解得到 r。直接方法是直接进行拟合，间接方法是构造投影方程进行求解。

5.14 PI WITH INDIRECT POLICY EVALUATION

使用间接策略评估进行策略迭代，之前讲过了，可以参照值迭代，教授没有多说，这里也不说太多了。

5.15 KEY QUESTIONS AND RESULTS

有两个比较重要的问题需要解决，第一个是投影方程是否有解，第二个是什么情况下映射 ΠT_μ 是压缩映射，ΠT_μ 是不是存在唯一的不动点。

假设现在有一个 single recurrent class and no transient states 的马尔可夫链，这个马尔可夫链的特点是状态转移概率不随时间变化，是一个固定的常数，同时所有状态都可能反复出现，同时给定了一个策略 μ，定义 $\xi_j = \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} P(i_k = j | i_0 = i) > 0$，此时 ξ_j 就是状态 j 的长期出现概率，也就是这个状态在马尔可夫链跳转的过程中出现的概率。

现在使用这个定义得到的 ξ 来做权重求加权欧几里得范数，即算子 Π，那

么可以知道：关于参数 α 和 ξ 的加权欧几里得范数定义的算子 ΠT_μ 是压缩算子；ΠT_μ 能够保证有不动点 Φr^* 且满足 $||J_\mu - \Phi r^*||_\xi \leq \frac{1}{\sqrt{1-\alpha^2}}||J_\mu - \Pi J_\mu||_\xi$

这个上界其实不重要，重要的是这种方法可以得到投影的采样分布，使用这个分布采样进行近似可以保证算法收敛，之前提到的发散的情况就不会出现了。

5.16 PRELIMINARIES: PROJECTION PROPERTIES

这一页来介绍一下为什么投影操作能够工作。使用投影 Π 在空间 S 内使用加权欧几里得范数进行投影有一个很重要的性质：勾股定理 (课件上是毕达哥拉斯定理)，即 $||J - \Phi r||_\xi^2 = ||J - \Pi J||_\xi^2 + ||\Pi J - \Phi r||_\xi^2$。

勾股定理生效可以知道投影具有非扩张性，也就是两个不同的成本函数 J 和 $\bar{J}$，使用算子 Π 对他们作用的时候满足关系：

$$||\Pi J - \Pi \bar{J}||_\xi \leq ||J - \bar{J}||_\xi, \forall J, \bar{J} \in \Re^n$$

使用勾股定理可以得到非扩张性：

$$||\Pi J - \Pi \bar{J}||_\xi^2 \leq ||\Pi J - \Pi \bar{J}||_\xi + ||(I - \Pi)(J - \bar{J})||_\xi^2 = ||J - \bar{J}||_\xi^2$$

5.17 PROOF OF CONTRACTION PROPERTY

这里来证明一下 ΠT_μ 是一个压缩映射。

首先给出一个定理：如果 P 是策略 μ 的状态转移矩阵，有

$$||Pz||_\xi \leq ||z||_\xi, z \in \Re^n$$

Proof 矩阵 P 中的元素记为 p_{ij}，对于所有 $z \in \Re^n$，由于 $\sum_{i=1}^n \xi_i p_{ij} = \xi_j$，有

$$||Pz||_\xi^2 = \sum_{i=1}^n \xi_i \left(\sum_{j=1}^n p_{ij} z_j \right)^2 \leq \sum_{i=1}^n \xi_i \sum_{j=1}^n p_{ij} z_j^2 = \sum_{j=1}^n \sum_{i=1}^n \xi_i p_{ij} z_j^2 = \sum_{j=1}^n \xi_j z_j^2 = ||z||_\xi^2$$

所以可以得到：$||Pz||_\xi \leq ||z||_\xi, z \in \Re^n$

证明完毕

$\sum_{i=1}^{n} \xi_i p_{ij} = \xi_j$ 可以参见上面关于 ξ_j 的定义：

$$\xi_j = \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} P(i_k = j | i_0 = i) > 0 \tag{5.1}$$

由于算子 Π 的非扩张性与策略成本函数 bellman 方程的定义，可以得到

$$||\Pi T_\mu J - \Pi T_\mu \bar{J}||_\xi \leq ||T_\mu J - T_\mu \bar{J}||_\xi = \alpha ||P(J - \bar{J})||_\xi \leq \alpha ||J - \bar{J}||_\xi$$

所以可以得到结论：ΠT_μ 是一个关于参数 α 的压缩映射。

5.18 PROOF OF ERROR BOUND

这一页要推导误差上界。假定 Φr^* 是算子 ΠT 的不动点，有

$$||J_\mu - \Phi r^*||_\xi \leq \frac{1}{\sqrt{1-\alpha^2}} ||J_\mu - \Pi J_\mu||_\xi$$

Proof 由毕达哥拉斯定理，有 $||J_\mu - \Phi r^{2*}||_\xi^2 = ||J_\mu - \Pi J_\mu||_\xi^2 + ||\Pi J_\mu - \Phi r^*||_\xi^2$
由于 J_μ 是算子 ΠT 的不动点，$J_\mu = \Pi T J_\mu$，有

$$||J_\mu - \Pi J_\mu||_\xi^2 + ||\Pi J_\mu - \Phi r^*||_\xi^2 = ||J_\mu - \Pi J_\mu||_\xi^2 + ||\Pi T J_\mu - \Pi T(\Phi r^*)||_\xi^2$$

由于算子 ΠT 是压缩算子，所以有

$$||J_\mu - \Pi J_\mu||_\xi^2 + ||\Pi T J_\mu - \Pi T(\Phi r^*)||_\xi^2 \leq ||J_\mu - \Pi J_\mu||_\xi^2 + \alpha^2 ||J_\mu - \Phi r^*||_\xi^2$$

因此，可以得到

$$\begin{aligned}
&||J_\mu - \Phi r^*||_\xi^2 \leq ||J_\mu - \Pi J_\mu||_\xi^2 + \alpha^2 ||J_\mu - \Phi r^*||_\xi^2 \\
\Rightarrow &(1-\alpha^2) ||J_\mu - \Phi r^*||_\xi^2 \leq ||J_\mu - \Pi J_\mu||_\xi^2 \\
\Rightarrow &||J_\mu - \Phi r^*||_\xi^2 \leq \frac{||J_\mu - \Pi J_\mu||_\xi^2}{(1-\alpha^2)} \\
\Rightarrow &||J_\mu - \Phi r^*||_\xi \leq \frac{||J_\mu - \Pi J_\mu||_\xi}{\sqrt{(1-\alpha^2)}}
\end{aligned}$$

证明完毕

5.19 MATRIX FORM OF PROJECTED EQUATION

给定了策略 μ，算子 T_μ，策略 μ 的状态分布概率矩阵 Φ 和成本向量 g，想要通过求解 $\Phi r = \Pi\xi T_\mu(\Phi r)$ 计算 r 的值。

首先，根据正交原理，$T_\mu(\Phi r)$ 到 Φr 的投影与近似子空间是正交的，更特殊地，Φr^* 和 $T_\mu(\Phi r^*)$ 的差值 (是一个 n 维向量) 和 Φ 产生的空间是正交的，也就是跟这个空间里所有的向量都正交，也就是说 $\Phi'\Xi\left(\Phi r^* - (g + \alpha P\Phi r^*)\right) = 0$。其中 Ξ 是一个由 $\xi_1, \xi_2, \cdots, \xi_n$ 构成的对角矩阵。

上面的等式可以写作 $Cr^* = d$，其中 $C = \Phi\Xi(I - \alpha P)$，$d = \Phi'\Xi g$，这个方程是一个 $s \times s$ 的方程，相对比较容易求解，但是计算 C 和 d 的过程比较困难，因为计算 C 和 d 的时候维度是 n，这个时候就可以使用仿真来计算他们的值了，使用仿真技术来估算大规模线性代数运算的计算结果，之前我们谈到过这个。

5.20 SOLUTION OF PROJECTED EQUATION

假设现在已经算出 C 和 d 了，需要得到 $r^* = C^{-1}d$，就需要计算 C^{-1}，如果 C 可逆，计算 C^{-1} 的计算量也很大，如果 C 不可逆，就没有办法这么求 r^* 了。

现在使用公式 $\Phi r_{k+1} = \Pi(g + \alpha P\Phi r_k)$，使用投影值迭代方法进行迭代求解，由于算子 ΠT 是一个压缩算子，所以迭代无穷次的时候一定会收敛于不动点 r^*。

投影值迭代要通过下述最小二乘表达式拟合 r：

$$r_{K+1} = \arg\min_{r\in\Re^s} ||\Phi r - (g + \alpha P\Phi r_k)||_\xi^2$$

令上式梯度等于 0，有 $\Phi'\Xi\left(\Phi r_{k+1} - (g + \alpha P\Phi r_k)\right) = 0$，可以得到迭代表达式 $r_{k+1} = r_k - (\Phi'\Xi\Phi)^{-1}(Cr_k - d)$

5.21 SIMULATION-BASED IMPLEMENTATIONS

上面的推导中，C 和 d 的维度很高含义计算所以可以使用仿真进行估计，然后使用估计的值进行计算，估计后直接使用 $\hat{r}_k = C_k^{-1}d_k$ 叫做 Least Squares Temporal Differences(LSTD)，估计后使用投影值迭代叫做 Least Squares Policy Evaluation(LSPE)，不管什么方法，主要的思路就是使用低维线性计算代替高维先行计算进行估计。

5.22 SIMULATION MECHANICS

采样机制就是使用马尔科夫链产生一个无穷维度的轨迹，观察状态转移 (i_t, i_{t+1})，提取 $\phi(i'_t)$ 和成本 $g(i_t, i_{t+1})$，这样就有了 k 个样本，利用这些样本使用如下公式估算 C 和 d:

$$d_k = \frac{1}{k+1}\sum_{t=0}^{k}\phi(i_t)g(i_t, i_{t+1}) \approx \sum_{i,j}\xi_i p_{ij}\phi(i)g(i,j) = \Phi'\Xi g = d$$

$$C_k = \frac{1}{k+1}\sum_{t=0}^{k}\phi(i_t)\left(\phi(i_t) - \alpha\phi(i_{t+1})\right)' \approx \Phi'\Xi(I-\alpha P)\Phi = C$$

如果是 LSPE 的话，使用 $G_k = \frac{1}{k+1}\sum_{t=0}^{k}\phi(i_t)\phi(i_t)' \approx \Phi'\Xi\Phi$ 来估算 G_k。

仿真机制进行近似能否收敛取决于采样数量，样本量越大，收敛性越强，近似效果越好，如果用另一种形式表达的话 (只是形式不不同，本质是一样的)，还可以写成 temporal differences(TD)。

5.23 OPTIMISTIC VERSIONS

仿真技术就是要通过样本估算要计算的变量的近似值，目前为止提到的 LSTD 和 LSPE 都是基于仿真的方法，它们可以经过比较少的计算给出想要的变量的近似值 (近似到什么程度取决于样本数量和近似方法)，我对于 OPTIMISTIC 的理解就是观察了一些样本，然后很乐观地认为这就是整个样本空间的样子，然后估算相应的变量值。

近似矩阵然后求逆的操作会导致误差，误差主要来源于采样噪声，导致近似矩阵有噪声，逆矩阵也带有噪声，噪声的影响主要取决于样本数量，样本数量越多，噪声影响越小，LSTD 就有这种问题。而 LSPE 不涉及这个问题，至少在噪声的问题上要好于 LSTD，因为 LSPE 本质上是一种迭代算法，一点一点地调整参数，积累样本，可以考虑在迭代公式前加一个比较小的系数 γ，用 $r_{k+1} = r_k - \gamma G_k\left(C_k r_k - d_k\right)$，可以理解为补偿，样本数量比较小的时候校正参数，衰减噪声的影响。

LSPE 这种采样迭代算法主要有两种实现，一种是采一个样本做一次计算，另一种是采一批样本做一次计算。

实际上，这种采样的计算方法会造成一些问题，实际应用的时候有很多技巧可以使用，可能会经过一些尝试才能得到比较好的效果。

6

LECTURE 5: Exploration and Aggregation

CONTENTS

6.1 DISCOUNTED MDP

折扣 MDP，略。

6.2 APPROXIMATE PI

近似策略迭代，使用近似参数 r 和特征矩阵 Φ 近似成本函数，细节略。

6.3 EVALUATION BY PROJECTED EQUATIONS

使用投影方程进行策略评估，原始表达式是 $\Phi r = T_\mu(\Phi r)$，由于系统状态数量 n 过多，计算难度很大，所以使用加权欧几里得范数算子 Π 进行投影，使用一部分状态来计算方程组的解代替算子 T_μ 下的不动点，即使用 ΠT_μ 的不动点作为最终的解。假设现在有一个策略，这个策略能够以平稳状态概率分布遍历马尔可夫链的所有状态，把这个平稳状态分布作为 ξ，此时 ΠT_μ 有唯一解 (ΠT_μ 是压缩映射)，同时有一些变种，比如 LSTD 和 LSPE。

6.4 EXPLORATION

通过对策略采集状态样本评估策略成本函数的时候，为了评估策略 μ 的成本，使用策略 μ 进行采样策略会倾向于访问它喜欢的状态，对于不经常访问的状态，在平稳状态分布 ξ 中的值几乎是 0，如果马尔可夫链不是可遍历的，这一部分状态的概率就真的是 0 了。估算相应的成本的时候，这些不具有代表性的状态对应的成本误差就会非常大，这个误差会很严重地影响策略改进得到的结果。也就是说，对当前策略不重要的状态可能对其他策略很重要，这些状态估值不准确会影响得到的策略的目标值。这个问题很严重，特别是系统状态转移随机性特别小的时候，比如一个没有噪声的确定性系统采样。为了解决这个问题，需要一边改变采样机制一边修改估值结果，换句话说就是我们需要找到一个探索能力更强的投影来构成投影方程，使用权重 $\zeta = (\zeta_1, \zeta_2, \cdots, \zeta_n)$ 代替 ξ，让采样时当前策略无关的状态出现得更频繁。

如果策略 μ 不是一个遍历性的策略，即使改变了采样权重，探索不足遇到的问题依然存在。

6.5 EXPLORATION MECHANISMS

关于采样机制，目前有两种思路解决探索不足的问题，一种是使用多个初始状态不同的短轨迹代替单个长轨迹，比如 geometric sampling 和 free-form sampling，短轨迹采样通过恰当地选择初始状态来丰富样本的多样性。

另一种方法是继续生成一条长轨迹，但是不使用待估值的策略生成样本，而是使用另一种策略生成样本，这种方法叫做 off-policy 方法。

6.6 POLICY ITERATION ISSUES: OSCILLATIONS

策略迭代除了探索，还有另一个很重要的问题，就是震荡。近似策略迭代不是确定能够收敛的，而是生成几个策略互相循环，也就是迭代到最后策略在几个策略间跳转。

课件中的图是近似参数 r 所在的空间，将参数空间划分成几个部分，每一个策略都对应一个划分，R_μ 是参数向量的集合，每一个 r 都对应一个近似成本函数 $\tilde{J}$，如果最小化 bellman 方程，就能够得到一个策略 μ，给定一个策略 μ，所有的 r 都可以记为 R_μ，也就是说 R_μ 中的每一个 r 经过策略改进之后都能得到策略 μ。

假设现在有一个策略 μ_k，通过求解投影方程得到相应的 r_k，现在这个 r_k 落在了子集 $R_{\mu^{k+1}}$ 中，也就是说使用 r_k 进行策略改进能够得到策略 μ^{k+1}，落在了子集 $R_{\mu^{k+2}}$ 中，继续估值和改进策略，这个循环就形成了。一般来说，如果策略评估是准确的，对于一个给定的策略 μ，权重向量会在这个闭环中一直循环下去，如果算法能够发现这个循环，那么这个循环持续几次之后就会停止。比较理想的情况是如果 r_μ 恰好在集合 R_{mu} 中，就不会循环，迭代会终止。

最好的情况是最后在不循环，或者在比较好的策略间循环，比较好的情况是在比较差的策略间循环，这样你可以发现问题，调整你的算法，最坏的情况是在好策略和坏策略间震荡，因为你不知道这个结果有多好，因为谁都不知道最优值是多少。

6.7 MORE ON OSCILLATIONS/CHATTERING

另一种情况是策略改进无法准确到达新策略，当前处于策略 μ_1，通过策略改进应该到达 μ_2，如果使用乐观算法 (optimistic policy iteration) 导致无法到达该有的新策略，然后策略估计估计的是不准确的策略的成本函数，然后继续策略改进无法准确到达新策略，就这样一直迭代下去，如果策略评价越来越乐观，震荡幅度就会越来越小。另一个比较奇怪的现象就是权重向量收敛到一个固定的值，但是策略可能还在震荡，因此不仅要检查权重向量，还要检查策略是否震荡，也就是说，权重向量收敛了，但是策略发散了，这种现象有时候会出现，但是很难分析成因。

从数学上说，震荡的本质是投影算子不具备单调性导致的现象，不具备单调性指的是有两个函数 J 和 J'，在近似子空间内对他们进行投影的时候，他们的大小关系与原关系相反 (不考虑存在偏序的情况)。课件之前对于算子 T 的单调性有过证明，但是如果把他们与投影算子结合，单调性就失去了，这也是震荡产生的根本原因。如果把 Π 换成一个单调算子 W，同时保证 WT_μ 也单调并且收缩，算法就不会出现震荡的现象，这个结论可以从震荡的数学角度进行解释。下面要讲一下如何避免震荡，引入状态聚合。

6.8 PROBLEM APPROXIMATION - AGGREGATION

求解动态规划的另一个方法是状态聚合，状态聚合是一种问题近似的方法，把问题化简，让问题更容易求解，然后就可以使用任意一种方法，精确算法或者近似算法都可以。如果定义 ϕ_{jy} 是原系统与聚合系统的状态转移概率，聚合系统的最优成本 $\hat{R}(y)$ 算出来以后就可以使用 $J^*(j) = \sum_y \phi_{jy}\hat{R}(y), \forall y$ 来计算原系统的成本函数，如果使用线性结构近似，ϕ_{jy} 也可以理解成状态 j 的特征。

6.9 HARD AGGREGATION EXAMPLE

硬聚合是一种比较简单的聚合方法，每个原始状态只对应一个聚合状态，见课件中的图，原系统有 1-9 一共 9 个状态，聚合生成 4 个聚合状态的聚合系统，Φ 是原系统的聚合概率，行表示状态的特征，第 i 行表示原状态 i 对应各聚合状态的概率，要求每行的所有概率加起来等于 1，列对应聚合状态与某原状态的情况，第 j 列表示聚合状态 j 对应各原状态的概率分布，某列的所有概率加起来就没有限制了，可能小于 1，可能等于 1，可能大于 1，可以根据他们的比例重新计算聚合状态到原装态的概率分布。

d_{xi} 表示聚合状态 x 到原状态 i 的概率，计算聚合状态 x_1 到聚合状态 x_2 的转移概率可以通过原系统的状态转移概率计算，可以使用 $\hat{p}_{x_1x_2} = \sum_{i,j=1}^{n} d_{x_1i}p_{ij}\phi_{jx_2}$ 来计算聚合状态 x_1 到聚合状态 x_2 的转移概率。

6.10 AGGREGATION/DISAGGREGATION PROBS

课件中的图给了聚合状态转移的过程，现在处于聚合状态 x_1，状态转移时现根据分解分布 d_{x_1i} 产生原始状态 i，再根据聚合分布 ϕ_{ix_2} 产生新的聚合状态 x_2，这样就可以进行状态转移或者采样了，同时，如果给定了两个分布 D 和 Φ，就可以进行聚合了。

6.11 AGGREGATE SYSTEM DESCRIPTION

聚合状态转移分布可以使用 $\hat{p}_{x_1x_2} = \sum_{i,j=1}^{n} d_{x_1i}p_{ij}\phi_{jx_2}$ 计算，矩阵形式可以表达为 $\hat{P}(u) = DP(u))\Phi$，同样，聚合系统的即时成本通过原系统的及时成本

与聚合分解分布计算 $\hat{g}(x,u) = \sum_{i,j=1}^{n} d_{xi} p_{ij}(u) g(i,u,j)$，矩阵形式可以表达为 $\hat{g} = DP(u)g$

6.12 AGGREGATE BELLMAN' S EQUATION

得到聚合系统的聚合状态转移概率 $\hat{p}$ 与聚合系统即时成本 $\hat{g}$，可以写出聚合系统的 bellman 方程 $\hat{R}(x) = \min_{u \in U} \left[\hat{g}(x,u) + \alpha \sum_{y} \hat{p}_{xy}(u) \hat{R}(y) \right], \forall x$，算出聚合系统的成本函数不动点 $\hat{R}$ 之后，可以使用 $\tilde{J}(j) = \sum_{y} \phi_{jy} \hat{R}(y), \forall j$ 来近似原系统的最优成本函数 J^*。

6.13 EXAMPLE I: HARD AGGREGATION

聚合的思路可以有很多，比如成本相近的原始状态聚合到一起，或者是状态相近的原始状态聚合到一起，如果原始系统的成本向量时聚合状态下的分段常值函数，这个硬聚合就是精确的，近似误差是 0。

软聚合和硬聚合的区别就是，硬聚合的 Φ 元素不是 0 就是 1，而软聚合有除了 0 和 1 之外的其他概率。

6.14 EXAMPLE II: FEATURE-BASED AGGREGATION

基于特征的聚合把原状态空间提取出的特征构成的特征空间进行聚合，然后把特征对应的原状态聚合成某个聚合状态，这种做法不需要离散化原状态空间，可以得到一个分段常数近似，一般来说，近似效果要好于直接离散化线性近似，因为基于特征的聚合是一个分段常数近似，是一个非线性近似方法，绝大部分非线性近似效果要优于线性近似，同时由于特征相似的原状态对应的成本也相似，被聚合成同一个聚合状态能够在降低维度的同时尽量少地避免近似精度丢失。

6.15 EXAMPLE III: REP. STATES/COARSE GRID

这页讲了另一个比较传统的方法，在一个连续状态空间中，取若干具有代表性的状态作为聚合状态，聚合状态转移从某个聚合状态 X 开始，使用原系统的状态转移矩阵转移到原系统状态 j，然后使用原状态-聚合状态的分布到达一个聚合状态，就完成了聚合状态与聚合状态之间的转移。这是一种比较好的离散化连续状态空间的方法，与邻域离散化状态的方法相比，这种以概率离散化的方法离散效果要更好，很适合于离散连续欧几里得空间，可以用来解决很多控制问题。

另一个大量应用的领域是部分可观测马尔可夫决策过程 (Partially Observable Markov Decision Process, POMDP)，POMDP 只能观测到一部分状态的信息，这样就可以定义一个在高维连续状态空间内的置信空间马尔可夫决策问题，将这个连续空间离散化，然后定义一个新的聚合状态转移概率，建模后就可以使用任意方法得到它的分段线性近似解。

POMDP 这部分没接触过，所以没能理解，只是原样放在这里了。

6.16 EXAMPLE IV: REPRESENTATIVE FEATURES

这一页的方法把前面讲的聚合方法综合到一起，不使用代表性状态而是代表性子集合，把特征相似的状态放到同一个集合里构成一个聚合状态，一个聚合状态表示原状态的一个子集合，聚合状态转移过程是从一个聚合状态开始，根据状态转移概率跳转到一个原始状态，再根据 ϕ 跳转到另一个聚合状态，这就是聚合状态之间的跳转过程。这种方法有一些限制，聚合状态子集合之间不可以相交，必须是互斥的；分解概率 d 只有在原始状态在聚合状态集合中的时候才有 $d > 0$，否则都等于 0；聚合状态概率 ϕ 只有在原始状态在聚合状态集合内的时候才等于 1，否则为 0，也就是说每一个状态最多对应唯一一个聚合状态集合。

前面的硬聚合是所有聚合状态的并集刚好是状态空间的情况，代表性状态聚合是每个聚合状态集合只有一个状态的情况。

6.17 APPROXIMATE PI BY AGGREGATION

在对原问题进行策略迭代的时候，可以使用聚合方法进行策略评估，也就是说通过 $R = DT_\mu(\Phi R)$ 来求解聚合状态对应的成本函数，然后使用 $\tilde{J} = \Phi R$ 来计算原状态的近似成本函数值，求 R 的过程可以使用仿真来完成。

如果使用投影算子 Π 代替聚合策略评估中的 ΦD，策略评估就变成了一个投影方程，看起来聚合策略评估和投影策略评估差不多，但是聚合策略评估有一个好处就是不会出现震荡的情况，而投影策略评估会导致震荡。

这种方法的限制就是概率分布 Φ 和 D 的概率分布是受到限制的，不可以使用任意分布进行计算。

7

LECTURE 6: Q-factor and some other issues

CONTENTS

7.1 LECTURE OUTLINE

这一次课主要讲了 Q 值相关的动态规划的内容，和其他方向的 ADP 还有一些话题。

1. Review of Q-factors and Bellman equations for Q-factors
2. VI and PI for Q-factors
3. Q-learning - Combination of VI and sampling
4. Q-learning and cost function approximation
5. Adaptive dynamic programming

6. Approximation in policy space
7. Additional topics

7.2 DISCOUNTED MDP

折扣 MDP 的回顾，略。

7.3 BELLMAN EQUATIONS FOR Q-FACTORS

Q 值的 bellman 方程回顾，略。

7.4 BELLMAN EQ FOR Q-FACTORS OF A POLICY

某策略下的 Q 值的 bellman 方程，略。

7.5 WHAT IS GOOD AND BAD ABOUT Q-FACTORS

使用 Q 值进行动态规划，精确算法和近似算法都可以用，之前讨论的性质和方法，压缩映射，收缩性，最优条件，收敛性，值迭代和策略迭代都可以使用 Q 值，包括那些近似理论也都可以用，比如投影方程，采样，探索，震荡，聚合等等。

关于 Q 值得一个很重要的事情是，如果现在有一个模型无关得控制器，可以计算所有状态-控制对的成本值，也就是 Q 值，控制的时候就不用 bellman 方程算控制了，可以直接在 Q 值中直接找最合适的控制去执行了。所以 Q 值主要的好处有两个：不需要模型，也就是转移概率，和计算控制的时候不需要使用 bellman 方程进行计算，可以直接构建模型最小化 Q 来寻找控制。

Q 值的坏处就是，搜索空间维度更高，存储空间和计算量需求更大。

7.6 Q-LEARNING

Q-learning 是一种采样形式的值迭代 (一种随机迭代算法)，运行过程是先使用采样机制 (可以是任何一种采样机制，概率机制或者确定性机制) 获得样本，然后使用迭代公式更新 Q 值，迭代公式中的迭代步长 γ_k 得非常小，一般与 $\frac{1}{k}$ 成正比。

$$Q_{k+1}(i_k, u_k) = (1 - \gamma_k)Q_k(i_k, u_k) + \gamma_k \left(g(i_k, u_k, j_k) + \alpha \min_{u' \in U(j_k)} Q_k(j_k, u') \right)$$

7.7 NOTES AND QUESTIONS ABOUT Q-LEARNING

Q-learning 工作的时候只需要一个模拟器，输入控制，输出系统状态和即时回报，一边采样一边更新 Q 值，这个操作也让 Q-learning 变成了一个异步算法，之前的内容讨论过异步算法，也就是不是所有状态的值都同时更新，上述操作每次只更新一个状态-控制对对应的值，所以这就是一个异步算法。

Q-learning 能工作，但是采样值迭代不能工作的原因是，一般的随机迭代算法工作时，通过采样计算成本的期望值，但是 bellman 方程有一个最小化操作的映射，如果对成本进行采样估算期望值，然后进行最小化，样本就不会工作。

采样值迭代不工作的原因教授是这么解释的，但是不是很理解，就原样放上来了，没有个人的理解和解释。

7.8 CONVERGENCE ASPECTS OF Q-LEARNING

Q-learning 收敛性证明比较复杂，涉及到随机近似理论和异步算法。从数学上讲，Q-learning 需要一个映射 F 把 Q 映射到 FQ，F 是一个带有 sup-norm 的压缩映射，为了让随机近似算法，压缩性很重要，由于有异步操作，sup-norm 也很重要。

一般性的随机近似算法，带有期望的一般性的不动点 $x = E_w \{f(x, w)\}$，想要找到以 w 为随机变量的映射 f 的期望值的不动点，假设 $E_w \{f(x, w)\}$ 是一个与 norm 相关的压缩映射，使用这个映射就可以通过 $x_{k+1} = E_w \{f(x, w)\}$ 不断地迭代得到一个不动点。

7.9 STOCH. APPROX. CONVERGENCE IDEAS

Q-learning 迭代的时候采集一个样本序列 $\{w_1, w_2\}$，使用这些样本估算 $x_{k+1} = E_w\{f(x_k, w)\}$，由于每次产生新样本进行估算的时候都需要使用当前有的样本进行一次平均计算，计算量非常大，为了解决这个问题，改变了迭代公式，使用 $x_{k+1} = (1-\gamma_k)x_k + \gamma_k f(x_k, w_k), k = 1, 2, \cdots$，如果 $\gamma_k = \frac{1}{k}$，那么这个新的迭代公式就和上面的那个迭代公式的估算结果是一样的。

Q-learning 的不动点计算公式 $Q = FQ$ 详细表达如下

$$(FQ)(i,u) = E_j\left\{g(i,u,j) + \alpha \min_{u'} Q(j,u')\right\}$$

7.10 Q-LEARNING COMBINED WITH OPTIMISTIC PI

Q-learning 的迭代公式 (如下) 在控制集合很大的时候计算量非常大。

$$Q_{k+1}(i_k,u_k) = (1-\gamma_k)Q_k(i_k,u_k) + \gamma_k\left(g(i_k,u_k,j_k) + \alpha \min_{u'\in U(j_k)} Q_k(j_k,u')\right)$$

为了减小计算量，可以维护一个策略 μ_k，每次需要求最小成本的时候直接使用这个策略计算控制，可以采用异步乐观策略迭代求这个策略，迭代过程中使用 $Q_{k+1} = F_{\mu^k}^{m_k} Q_k$ 进行策略评估，使用 $\mu^{k+1}(i) \in \arg\min_{u\in U(i)} Q_{k+1}(i,u)$。实际上这种方法被事实证明不工作 [12, 13](没有找到具体的内容，只能搜得到 93 年他俩合作的文章)，不过可以通过很小的改进来解决这个问题 [7]。

7.11 Q-FACTOR APPROXIMATIONS

大规模精确 Q-learning 由于存储空间有限和维度过大无法使用，所以考虑使用基函数近似来解决这个问题，使用状态 i 和控制 u 的特征向量函数 $\phi'(i,u)$ 乘以权重向量 r 来近似 Q 值。在使用样本近似策略迭代搜索 r 的时候，从某个策略出发，收集若干样本 (可能是几十个也可能是几百个，还有可能只有一个)，评价策略，可以使用之前讲过的 LSTD 和 LSPE 之类的方法进行策略评价，然后使用策略评价的结果进行策略迭代，实际操作中人们通常使用乐观策略迭代来求解权重向量。这种算法比较混乱，有的与理论相关，有的与有效性相关，总之没有一个很明确的性质和结论，有效性与震荡表现之类的东西，很复杂，但是使用得很广泛，得到的评价也都是这个算法很成功。

举个例子，这种算法在最优停止问题上很有效，就是每一个阶段只有两个控制，继续或者停止，比如金融操作，卖出去或者留着，在这一类问题上表现非常好

7.12 BELLMAN EQUATION ERROR APPROACH

这一页要讨论的是另一种能够代替投影方法的近似算法-bellman 误差法。给定一个策略 μ，使用线性结构 $\tilde{Q}_\mu(i,u;r_\mu)=\phi(i,u)'r_\mu$ 近似 Q 值，用最小化 bellman 误差的方法来求当前近似效果最好的 $r_\mu \in \arg\min_r ||\Phi r - F_\mu(\Phi r)||_\xi^2$，其中 $||\cdot||_\xi$ 表示使用权重 ξ 得到的欧几里得范数。这是一个最小二乘问题，给定一个策略，求解这个最小二乘问题找到 r_μ，然后使用这个 r_μ 进行策略改进，然后再最小化 bellman 误差求解新的 r，这么迭代下去。

对于随机问题，这种方法更适合确定性问题，因为对于一个确定性问题，不存在转移概率，状态转移过程中能得到一个确定的新状态，生成样本之后，就可以用这些样本计算最小二乘问题，如果使用所有状态求解 r，近似结果会很精确，误差也会很小，如果没有用所有状态进行计算，能够采集具有代表性的样本进行计算也能够得到一个误差比较小的近似权重向量。这是一个线性的最小二乘问题，求解的方法很多，但是必须要保证矩阵可逆，方法是采集足够的样本，样本数量不足的时候是没有办法保证矩阵可逆的。

随机问题与确定性问题相似但是更复杂，因为随机问题每个阶段都需要计算 Q 值，所以说随机性问题更复杂 (不太理解这个地方说的每个阶段都需要计算 Q 值到底是指什么，个人的理解是在计算 bellman 误差的时候，由于转移概率的存在，新状态 Q 值不能只用采集到的样本进行计算，需要计算新状态 (不止采集到的样本) 的 Q 值的期望，这个地方会产生比较大的计算量)。

7.13 LINEAR-QUADRATIC PROBLEM

谈论自适应动态规划之前要先回顾一下之前讲过的东西，系统状态转移方程 $x_{k+1}=Ax_k+Bu_k, x_k\in\Re^n, u_k\in\Re^m$，总成本 $\sum_{k=0}^{\infty}\left(x_k'Qx_k+u_k'Ru_k\right), Q\geq 0, R>0$，$R$ 一定要是一个正定矩阵，最优控制 $\mu^*(x)=Lx$，对于某个策略 μ，Q 值可以表示为一个二次项表达式

$$Q_\mu(x,u)=\begin{pmatrix}x' & u'\end{pmatrix}K_\mu\begin{pmatrix}x\\ u\end{pmatrix} \tag{7.1}$$

现在假设 $A\in\Re^{n\times n}$ 和 $B\in\Re^{n\times m}$ 是未知的，即该系统是一个时间无关的离散时间线性系统，基函数有若干元素，即 $x^ix^j,\quad u^iu^j,\quad x^iu^j,\quad \forall i,j$，这三个基函数也是矩阵 Φ 的一行 $\phi(x,u)'$。

线性策略 μ 对应的 Q 值 Q_μ 可以使用 $Q_\mu(x,u)=\phi(x,u)'r_\mu$ 精确地表达，其中 r_μ 是由 7.1中的 K_μ 构成的。

7.14 PI FOR LINEAR-QUADRATIC PROBLEM

然后就可以使用 $\min_r \sum_{x_k,u_k} |\phi(x_k,u_k)'r-\left(x_k'Qx_k+u_k'Ru_k+\phi\left(x_{k+1},\mu(x_{k+1})\right)'r\right)|^2$ 和 $\bar{\mu}(x)\in\arg\min_u\left(\phi(x,u)'r_u\right)$ 分别进行策略评估和策略改进。

这种方法可以处理连续空间和线性/非线性系统在连续/离散时间下的控制问题。

7.15 APPROXIMATION IN POLICY SPACE

略过。

7.16 APPROXIMATION IN POLICY SPACE METHODS

使用近似权重向量 $r=(r_1,r_2,\cdots,r_s)$ 来参数化策略，直接根据状态 i 与近似参数 r 使用 $\mu(i,r)$ 计算控制。每一个 r 都对应一个策略 $\tilde{\mu}(r)=\{\tilde{\mu}(i;r)|i=1,2,\cdots,n\}$，也有相应的成本向量 $J_{\tilde{\mu}(r)}$，现在这个问题就是使用各种方法，比如随机搜索，梯度方法或者其他方法搜索 r 来最小化累加投影成本 $\sum_{i=1}^n \xi_i J_{\tilde{\mu}(r)}(i)$，其中 $\xi=(\xi_1,\xi_2,\cdots,\xi_n)$ 是依赖于状态的权重，根据之前的讨论，也可以理解为这些状态的概率分布，如果状态数量过多无法全部访问，可以使用这个分布进行采样。

上述是直接参数化策略的方法，还有间接参数化策略，即使用如下表达式近似：

$$\tilde{\mu}(i;r)=\arg\min_{u\in U(i)}\sum_{j=1}^n p_{ij}(u)\left(g(i,u,j)+\alpha V(j;r)\right),\forall i$$

这种参数化的间接近似方法优势在于如果你知道成本向量的效果好的近似结构的时候，就可以使用成本的近似来间接的策略近似，就不用再去找策略比较好的特征了，可以说是一种通用的近似方法了。

策略近似被叫做 actor，成本值近似叫做 critic。

7.17 COMBINATION WITH APPROXIMATE PI

求解近似策略的近似参数 r 有很多种方法，首先是随机搜索方法，这种方法很直接而且在很多领域都很成功，比如俄罗斯方块。随机搜索方法的工作原理是在一个给定的近似参数 r 邻域随机生成若干新的近似参数 r'，找到一个更好的近似参数。不管搜索空间是连续的还是离散的，这种方法都能使用。

另一种方法是基于梯度的方法 (比如策略梯度)，同样被很广泛地应用，工作得原理是计算 $\sum_{i=1}^{n} \xi_i J_{\bar{\mu}(r)}(i)$ 关于 r 的梯度，然后沿着这个梯度搜索新的 r，策略梯度方法也可以使用仿真采集样本，然后根据采集得到的样本计算梯度进行梯度下降，有一个问题就是仿真德噪声会对算法造成很大的影响。然而梯度方法有很多局限性也有很多坏处，可能很成功也可能完全不工作，算法可能非常慢，难以收敛，也可能完全没法预判运行效果，不过很容易实现，所以应用很广泛。在很多文献中，基于梯度的方法都可以从理论上保证收敛，但是实际应用的时候很多并不能真的收敛

7.18 COMBINATION WITH APPROXIMATE PI

在使用策略近似策略迭代的时候，算法是这么工作的，首先使用近似值迭代方法评估当前的近似策略，然后使用优化算法 (比如之前提到的随机搜索或者梯度方法) 优化策略的近似参数，然后再评估，一直这么迭代下去。

下面的公式与课件不一样，是个人理解，不能保证对。

同时使用值近似和策略近似进行策略迭代的时候，目的是最小化累计成本

$$\min_{r_c, r_a} \sum_{i=1}^{n} \xi_i \sum_{j=1}^{n} p_{ij}\left(\bar{\mu}(i; r_a)\right) \left(\tilde{J}_{\bar{\mu}}(j, \bar{\mu}; r_c)\right)$$

其中 r_c 和 r_a 分别为 critic 和 actor 的近似参数，$\tilde{J}_{\bar{\mu}}(j, \bar{\mu}; r_c)$ 为状态 j 下使用策略 $\bar{\mu}$ 的近似成本，$\bar{\mu}$ 也做参数的原因是如果需要的话，可以使用 $\bar{\mu}$ 进行采样，既可以近似状态成本也可以近似状态-动作对的成本。

迭代步骤是先评估策略，求近似效果最好的 r_c，然后求当前策略评估参数下能最小化成本的策略近似参数 r_a，然后进行新一轮迭代，策略评估策略改进一直进行下去，策略评估和策略迭代的方法前面已经谈过很多了，这里就不详细说了。

7.19 TOPICS THAT WE HAVE NOT COVERED

这一页是我们没有讲，但是也很有用的内容。

1. 折扣半马尔科夫决策过程，随机最短路径问题，平均成本问题，回合制游戏；
2. 连续空间控制问题；
3. 连续时间问题；
4. 自适应动态规划，连续时间确定性最优控制问题，成本函数的导数近似和成本函数的差分近似；
5. 近似策略评估与近似策略的随机搜索方法；
6. 基函数自适应方法 (自动产生最好的基函数)；
7. 基于仿真的一般性线性问题。

7.20 CONCLUDING REMARKS

关于 ADP，这一页做了一些总结性的东西。

1. 使用 ADP 算法，没有赢家，没有最好的算法，如果想要找到一个最好的算法，是找不到的，ADP 算法都需要根据要解决的问题进行设计，需要理解 ADP 算法的本质；
2. ADP 算法有很多有趣的理论，同时需要知道，ADP 算法得不到严格的性能的保证，解决问题的时候尝试一种方法，但是这种方法很可能不工作，这时候就需要多尝试几种方法或者是对某个方法进行改进甚至是创造一个新的方法来解决问题；
3. 所有的方法都有各自的优势和缺陷，使用投影方程进行近似策略迭代会出现震荡现象，聚合问题中近似结构会受到限制，也就是说基函数需要按照概率分布来进行设计，否则算法很容易不工作，近似策略方法很容易实现，但是很容易不工作，不管优化策略近似参数的时候使用基于仿真的梯度方法或者随机搜索什么的，都不可靠，可能不工作；
4. 尽管这些缺陷导致这些算法不好用，但是他们在非常复杂的问题中得到了很好的应用，表现也非常好，而且这些问题通常没有其他更好的方法能使用了；
5. 我们将过了近似策略迭代，近似值迭代，Q-learning，不同类型的基于 Q 值的算法，但是还有很多其他方法我们没有提到，比如 rollout 方法就是很简单很成功而且可靠性很高的方法，这个方法不是很消耗资源，但是非常可靠而且能给出很好的结果，使用线性结构来近似一个确定性的 bellman 线性方程是一种很成功并且受到很多关注和应用的方法，这个方法值得关注；
6. 虽然 ADP 是一个试错的领域，但是使用 ADP 解决问题的时候，能够理解相应的理论知识很重要，需要挖掘理论知识来指导应用；

7. 对于理论性知识，能够把这些理论实践是一个非常大的挑战，使用 ADP 算法的对创造力提出了很高的要求。

Bibliography

[1] Dimitri P Bertsekas. Neuro-dynamic programming. In *Encyclopedia of optimization*, pages 2555–2560. Springer, 2008.

[2] Dimitri P Bertsekas. *Abstract dynamic programming*. Athena Scientific Belmont, MA, 2013.

[3] Dimitri P. Bertsekas. *APPROXIMATE DYNAMIC PROGRAMMING A SERIES OF LECTURES GIVEN AT TSINGHUA UNIVERSITY JUNE 2014*. video lectures at homepage of Bertsekas, 2014.

[4] Dimitri P. Bertsekas. *Feature-Based Aggregation and Deep Reinforcement Learning: A Survey and Some New Implementations*. arXiv:1804.04577, 2018.

[5] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 2005.

[6] Dimitri P Bertsekas and Huizhen Yu. Projected equation methods for approximate solution of large linear systems. *Journal of Computational and Applied Mathematics*, 227(1):27–50, 2009.

[7] Dimitri P Bertsekas and Huizhen Yu. Q-learning and enhanced policy iteration in discounted dynamic programming. *Mathematics of Operations Research*, 37(1):66–94, 2012.

[8] Justin A. Boyan. Least-squares temporal difference learning. In *ICML*, 1999.

[9] Justin A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246, Nov 2002.

[10] A Nedić and Dimitri P Bertsekas. Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, 13(1-2):79–110, 2003.

[11] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988.

[12] Ronald J Williams and Leemon C Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical report, Citeseer, 1993.

[13] Ronald J Williams and Leemon C Baird III. Analysis of some incremental variants of policy iteration: First steps toward understanding actor-critic learning systems. Technical report, Tech. rep. NU-CCS-93-11, Northeastern University, College of Computer Science, Boston, MA, 1993.