BERTSEKAS 2014 年在清华的 ADP 课程笔记

史丰源

March 11, 2019

Contents

1 Introduction **2**

2 动态规划能求解的问题与基本概念和符号 **2**
2.1 动态规划能解决的问题 2
2.2 动态系统的数学描述 3
2.2.1 离散时间动态系统 3
2.2.2 连续时间动态系统 4
2.3 控制策略 μ 5
2.3.1 平稳控制策略 5
2.3.2 非平稳控制策略 5
2.4 动态规划问题成本函数的数学描述 5
2.4.1 非折扣问题 6
2.4.2 折扣问题 6
2.5 动态规划问题与更简洁的表达方式 7

3 精确动态规划算法 **7**
3.1 精确值迭代算法 9
3.2 精确策略迭代算法 13

4 近似动态规划 **16**
4.1 求解中可能用到的一些内容 17
4.1.1 仿真 17
4.1.2 ROLLOUT 算法 17
4.1.3 OPTIMISTIC POLICY ITERATION 18
4.1.4 同步与异步算法 18
4.2 值空间近似 20
4.2.1 值空间近似的一般思路 20
4.2.2 拟合值迭代 22
4.2.3 投影值近似 23
4.3 近似策略迭代 24
4.3.1 近似策略迭代的性质 26
4.3.2 近似策略迭代的探索机制与震荡现象 28
4.4 问题近似：状态聚合 30

1 Introduction

BERTSEKAS 2014 年在清华给了一个关于 ADP 的暑期课程，这个课程是教授关于 ADP 的工作的综合性的课程，包括 “Neuro-Dynamic Programming” [2]，“Dynamic Programming and Optimal Control, Vol. II: Approximate Dynamic Programming” [3] 和 “Abstract Dynamic Programming” [4]，还可以访问BERTSEKAS 的主页获取更多资料，课程的主旨是如何使用各种各样的近似手段来降低动态规划的求解难度，然后使用合适的 ADP 算法来解决相应的问题，课程先给了精确动态规划的算法，值迭代与策略迭代，然后给出了收敛性与最优性的证明，然后给出了动态规划的速记符号，一种更简洁的数学表达形式，然后介绍了各种各样的近似动态规划的近似手段，课程主要面对的是具有动态规划基础的科研工作者与需要使用动态规划解决问题的工程师，这是一个总结性的完整的笔记，整理笔记的时候没有添加动态规划的基础介绍部分，阅读此笔记同样需要对动态规划有基本的理解。

Bellman 提出了动态规划 (Dynamic Programming) 方法 [5] 并给出 “维数灾” (Courses of Dimensionality) 的概念，Powell 认为维数灾有三种 [7]，动作空间过大，状态空间过大和随机干扰，这三者共同构成了维数灾，导致动态规划失效。维数灾是问题规模过大导致动态规划求解困难，动态规划的另一个限制是系统不可知，即系统的状态转移概率未知，同样会限制动态规划的使用，很多问题都是实时性问题，即要求立刻给出一个决策用以执行，动态规划算法如果从新状态产生开始计算，虽然能够得到最优解，但是耗费时间非常长。动态规划不止能处理动态问题，只要能够构造出动态规划规定的模型，即使静态问题也可以使用动态规划求解，上面的这些困难综合在一起，很多研究人员就有了足够的动机采用近似手段让动态规划能够给出一个可接受的解，也就是近似动态规划 (Approximate Dynamic Programming)，ADP 在不同的领域中都得到了相应的应用，主要有控制理论，人工智能与运筹学，ADP 在不同领域也有不同的名字，控制领域中叫它自适应动态规划 (Adaptive Dynamic Programming)，运筹学领域还叫近似动态规划，人工智能领域叫他强化学习 (Reinforcement Learning)，，不同领域关注的重点不一样，但是本质还是一样的，都是发源于动态规划，采用近似手段的算法。

2 动态规划能求解的问题与基本概念和符号

2.1 动态规划能解决的问题

很多领域的很多问题最终都会产生不同形式的优化问题，求解这个优化问题也就等同于解决了最初的问题，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。不确定性问题可以分为平稳系统 (stationary policy) 和非平稳系统 (nonstationary policy)，平稳系统的特点是系统方程的状态转移概率分布不随着时间改变，非平稳系统的特点是系统方程的状态转移概率分布随着时间改变，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。

对于平稳系统还有一个性质，即如果 g 有上界 M，则平稳系统有 $|J_\pi(x)| \leq \frac{M}{1-\alpha}$，证明会在下面折扣系统中给出。

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 (平稳系统与非平稳系统) 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。

2.2 动态系统的数学描述

2.2.1 离散时间动态系统

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，或者离

散时间点的数量，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(可能是连续变量也可能是离散变量)，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。相应地，无限期问题 ($N\to\infty$) 的总成本表达式为 $E\left\{g_\infty(x_\infty)+\sum_{k=0}^{\infty} g_k(x_k,u_k,w_k)\right\}$

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，此时系统总成本为 $E\left\{g_N(x_N)+\sum_{k=0}^{N-1} P(x_{k+1}|x_k,u_k)g_k(x_k,u_k)\right\}$。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。

2.2.2 连续时间动态系统

有一个系统，系统方程为 $dx(t)/dt=f(x(t),u(t),w(t))$，成本函数为 $J(t_0)=E\left\{\int_{t_0}^{T} g(x(t),u(t),w(t))dt\right\}$，其中 t 为系统当前时间，T 为系统运行总时间，如果 $T\to\infty$，成本函数表达为 $J(t_0)=E\left\{\int_{t_0}^{\infty} g(x(t),u(t),w(t))dt\right\}$。

2.3 控制策略 μ

定义控制策略 μ，有 $u = \mu(x)$，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，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，控制策略有两种，平稳控制策略 (stationary policy) 和非平稳控制策略 (nonstationary policy)。

2.3.1 平稳控制策略

平稳策略 μ，是一种时间无关的策略，不管当前处于系统演化的哪一个阶段，都可以使用同一个控制策略进行决策，另一种是非平稳策略 μ_k 或者 $\mu(t)$，即时间相关策略，不同的决策阶段使用不同的控制策略，离散时间系统使用策略 μ_k，连续时间系统使用策略 $\mu(t)$。平稳控制策略一般在无限期问题中使用，由于无限期问题中控制的时间终点都是不存在的，因此两个不同时间开始的系统的控制成本极限值是相等的，所以无限期问题中平稳控制策略用得比较多，但是如果把系统当前时间作为系统状态的一部分，有限期系统也可以使用平稳控制策略。以离散控制空间中使用平稳控制策略 μ 的离散时间系统为例，控制策略 μ 对应的总成本函数为 $E\left\{g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu(x_k), w_k)\right\}$。

2.3.2 非平稳控制策略

非平稳控制策略只能在有限期系统中使用，因为如果在无限期问题中使用非平稳控制系统，就需要搜索无数个策略，显然无法求解。

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。

2.4 动态规划问题成本函数的数学描述

动态规划问题按照不同的分类标准可以分成不同的类别，按照采样时间分，可以分成离散时间系统和连续时间系统；按照系统的状态转移概率分，可以分成不确定性系统和确定性系统，其中不确定性系统根据状态转移概率是否时间相关还分为平稳系统和非平稳系统；按照系统方程是否是线性分，可以分为线性系统和非线性系统。其实这种分类方法是按照状态方程分类，下面要从成本函数的角度进行分类，按照控制过程中观察到的成本是否随着时间衰减分，可以分为折扣问题和非折扣问题。

2.4.1 非折扣问题

不关心系统的类别,控制系统过程中产生的总成本为 $E\left\{g_N(x_N)+\sum_{k=0}^{N-1}g_k(x_k,u_k,w_k)\right\}$，每次控制后观察到的即时成本为 $g(x,u,w)$，不加处理地把它们累加起来作为总成本，最小化这个总成本，这个优化问题就是一个非折扣问题。

非折扣问题只能出现于有限期问题中 [1]，原因是如果在无限期问题中使用非折扣方式计算总成本会出现总成本无界的现象从而导致问题无解，对于无限期问题比较好的处理方式就是使用折扣方法让总成本有界。

2.4.2 折扣问题

每次控制后观察到即时成本 $g(x,u,w)$，定义一个参数 $\alpha\in[0,1)$，使用 $E\left\{g_N(x_N)+\sum_{k=0}^{N-1}\alpha^k g_k(x_k,u_k,w_k)\right\}$ 计算总成本，由于除了当前的即时成本之外，后续的即时成本都受到了衰减，优化问题的目标函数所有项都打了折扣，把这个优化问题叫做折扣问题。

对于一个平稳系统，折扣问题的成本值有上界 $\frac{M}{1-\alpha}$，如果 $|g|\le M$，证明如下:

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。

2.5 动态规划问题与更简洁的表达方式

对于一个最小化问题，动态规划的本质是求解一个线性方程组，也就是 bellman 方程，以离散状态空间离散决策空间内的无限期离散时间系统为例，有 bellman 方程 $J(x_0) = \min\limits_{u_t, t\in[0,\infty]} \lim\limits_{N\to\infty} E\left\{\sum\limits_{t=0}^{N-1} \alpha^t g_t(x_t, u_t, w_t)\right\}$，目标是找到最优决策序列 $\pi = (u_1, u_2, \cdots, u_N)$，更一般地，有 bellman 递归式 $J(x_k) = \min\limits_{u_k} E\left\{g_k(x_k, u_k, w_k) + \alpha J(f(x_k, u_k, w_k))\right\}$，引入算子 T 并且重新定义成本函数 $(TJ)(x_k) = \min\limits_{u_k} E\left\{g_k(x_k, u_k, w_k) + \alpha J(f(x_k, u_k, w_k))\right\}$，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 $(T_\mu J)(x_k) = E\left\{g_k(x_k, \mu(x_k), w_k) + \alpha J(f(x_k, \mu(x_k), w_k))\right\}$，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。

3 精确动态规划算法

比较常见的动态规划算法，比如教材上给出的有向无环图的单源点最短路径的动态规划算法都是从最后一个状态对应的成本算起，一步一步向前退，这种算法是可以保证最优性的，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

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。

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** $\pi^* = \left\{\mu_0^*, \mu_1^*, \cdots, \mu_{N-1}^*\right\}$

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。

上面的算法是从最后一个状态出发一点一点往前推，如果换一个角度，使用迭代求解动态规划，主要有两种动态规划算法，值迭代和策略迭代，现在先聊一聊精确的值迭代和策略迭代。

3.1 精确值迭代算法

离散状态系统中有状态集合 X，系统状态 $x\in X$，动态规划算法实际是想求所有状态对应的成本向量 $J(x)$ 的值，bellman 方程组中各方程可以表达为 $J(x_k)=\min_{u_k} E\left\{g_k(x_k,u_k,w_k)+\alpha J(f(x_k,u_k,w_k))\right\}$，如果使用迭代算法，从一个初始成本向量 J_0 出发，经过迭代后最终收敛于最优成本向量 J^*，这个最优的成本向量 J^* 就是 bellman 最优算子 T(与上面介绍过的简洁的表达形式 T 相同) 的不动点，也就是说，从 J_0 出发，不断地对 J_0 使用算子 T，最终会收敛于唯一的不动点 J^*，即

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

证明过程如下：

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 $\lim\limits_{M,N\to\infty} d(f^M(x), f^N(x)) = 0$，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。

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 $d(\mu_1, \mu_2) = \max_{s\in\mathcal{S}} |(||a_1(s) - a_2(s)||_2)|$

bellman 最优方程 (值迭代) 的矩阵形式迭代式:

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

bellman 期望方程向量形式为 $\mathbf{u}_{new} = T^*(\mathbf{u}) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{u}$，其中 $\mathcal{P}^a$ 是控制 a 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^* 后新的点距离为

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

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

所以 T^* 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 $\mathbf{v}^*$ 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 $\mathbf{v}^*$

值迭代最优性，即 bellman 最优方程的解，也就是最优解一定 J^* 满足方程 $J^* = TJ^*$，证明如下：

Proof 由上面内容可知，$J^*(x) = (T^k J_0) + \frac{\alpha^k M}{1-\alpha}$，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：

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

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

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

所以有 $J^* = TJ^*$

证明完毕

3.2 精确策略迭代算法

值迭代的目的是求各状态对应的成本值，另一种方法是求最优策略，使用策略的好处前面已经分析过了，策略迭代算法分成两步：策略评估和策略改进，策略评估是给定一个策略，计算该策略下各状态对应的成本值，得到这个成本值之后再使用策略改进找到一个更好的策略，然后继续进行策略评估-策略改进的迭代直到迭代收敛，得到一个最优策略，算法的工作过程是这样的：

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 J_{μ^k}，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-改进的循环直到收敛，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。

策略迭代收敛性证明如下：

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

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

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 $\mathbf{v}_\pi^*$

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

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

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^*

策略迭代最优性证明如下：

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

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

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

已知 $|g| \leq M$，所以上式满足

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

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 $N \leq M$ 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

4 近似动态规划

由于动态规划有各种各样的局限性，计算量过大，存储需求过大，模型可能未知，不管是控制，运筹还是人工智能都无法直接应用，控制领域 (一般用 DP 解决最优控制问题) 的火箭着陆问题，运筹学的VRP问题和人工智能的游戏 (比如flappy bird) 都是教科书上的例题，解决现实生活中的问题的时候 DP 移然无法使用，所以人们有了足够的动机来开发各种近似手段，目前主要有三种近似

手段：问题近似，值近似和策略近似，本文的安排也按照这三部分安排，还包括了一些求解的小技巧。

ADP 部分的安排是这样的，先把求解会用到的小技巧做一个介绍，然后是对值的近似，接下来是对问题的近似，最后简单提一下对策略的近似 (实际上策略近似很好用，但是教授的课件没有重点讲，同样只是提了一下，忠于课程的原则，这里也不大篇幅地讨论策略近似了)。

4.1 求解中可能用到的一些内容

4.1.1 仿真

前面说过，模型未知 (主要是状态转移概率未知，实际上除了控制范围和维度与系统状态维度可以什么都不知道) 会导致 DP 失效，也就无法建立 MDP 模型，这个时候就必须使用仿真技术，仿真技术其实只是模拟马尔可夫链跳转，观察新的系统状态和即时成本的方式，然后使用仿真技术观察到的这些信息估算系统的 J 值或者 Q 值。另一种情况就是需要进行大量计算的时候，必须计算向量内积，计算积分 (常用蒙特卡洛法) 等等，使用数学手段需要的时间非常长 (有些存储空间以 GB 计的矩阵) 这时候就可以使用仿真手段来求解。

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，也就是说，算法不知道这个问题的模型是什么，但是人要知道问题模型是什么才能写得出模拟器，如果人也不知道模型是什么，那就只能通过物理采样获得数据进行求解，当然求解效率不会很高，实际上，这种情况不管用什么方法得到的解质量都不会很高，甚至面临问题不可解决的困境。

4.1.2 ROLLOUT 算法

Stuuon 的 Reinforcement Learning:An Introduction(second edition)的 8.10 节给了介绍，简单地说，就是在当前状态基于某种策略对不同的控制进行大量的采样，然后使用采样结果估计这些控制的成本，再选择控制进入下一阶段，这些估值不会被保留，只是采样过程中用到的一种选择控制的方法。

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。

4.1.3 OPTIMISTIC POLICY ITERATION

没有找到合适的中文翻译，直译为乐观策略迭代，总感觉很别扭，就不用中文表达了。

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。

4.1.4 同步与异步算法

上面的精确 VI 和 PI 迭代过程中每次迭代都同时对所有成本函数进行更新，这种更新方式叫做同步更新，同步更新的算法叫做同步算法，如果不是所有成本都是同步更新的，就叫做异步算法。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。

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)

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^*。

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

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

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。目前就笔者所知道的比较好的工作中，[6] 就是基于这种思想的，虽然还是受到各种各样的限制，比如参数敏感，异步机数量敏感 (异步机数量过多的时候完全无法收敛) 但是还是有自己的优势的。

4.2 值空间近似

4.2.1 值空间近似的一般思路

由于 J 值和 Q 值没有本质区别，所以值空间近似主要在 J 值近似的基础上展开，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 J 函数，有些时候还可以通过近似成本函数 (Q 函数或者 J 函数) 的差分求解问题

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。

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。

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 $\Re^{(1+q+q^2)}$，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。

4.2.2 拟合值迭代

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。

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，也就是做一次参数拟合，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。

4.2.3 投影值近似

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：$\min_{r \in \Re^s} \sum_{t=1}^{k} \left(\phi((i_t)'r) - J(i_t)\right)^2$，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。

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：$\max_i |\tilde{J}_{k+1}(i) - T\tilde{J}_k(i)|$，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，但是下面会给出一个失败的例子。

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 $J(i)=\min\limits_{u}\sum\limits_{j=1}^{n}p_{ij}(u)(g(i,u,j)+\alpha\tilde{J}(j),\forall i$，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。

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。

4.3 近似策略迭代

原始的精确策略迭代有两步，策略评估和策略改进，下面这一段用到了还没有介绍的内容，如果遇到不理解的内容，继续看下去，投影，震荡，探索什么的内容都会在下面给出。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。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。

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。

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

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。

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。

4.3.1 近似策略迭代的性质

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。

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。

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。

投影算子使用之后策略迭代能够工作，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：

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

下面来证明一下 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：

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

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。
投影策略迭代的压缩映射推导过程如下：
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

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

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。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 $r_{k+1} = r_k - (\Phi'\Xi\Phi)^{-1}(Cr_k - d)$

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。

4.3.2 近似策略迭代的探索机制与震荡现象

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)。

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 $r_{k+1} = r_k - \gamma G_k (C_k r_k - d_k)$，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。

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。

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。

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_μ 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。

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。

4.4 问题近似：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。

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。

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。

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 $\hat{g} = DP(u)g$

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^*。

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。

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。

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。

5 一点闲话

感谢看到这里的朋友们，这份文档是整理了整个课程的所有笔记，六天的课程讲得有点乱，由于时间和篇幅关系整理得也不是很好，条理还是不够清晰，说实话，这个文档要是别人写的，我可能根本看不下来，再次感谢愿意花费这么长时间耐心看下来的朋友们。

References

[1] Andrew G Barto and Richard S Sutton. Reinforcement learning: An introduction. 1998.

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

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

[4] 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.

[5] Stuart Dreyfus. Richard bellman on the birth of dynamic programming. *Operations Research*, 50(1):48–51, 2002.

[6] Volodymyr Mnih, Adri¨¤ Puigdom¨¨nech Badia, Mehdi Mirza, Alex Graves, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 2016.

[7] Walter A. Shewhart and Samuel S. Wilks. *Approximate Dynamic Programming: Solving the Curses of Dimensionality, Second Edition*. 2011.