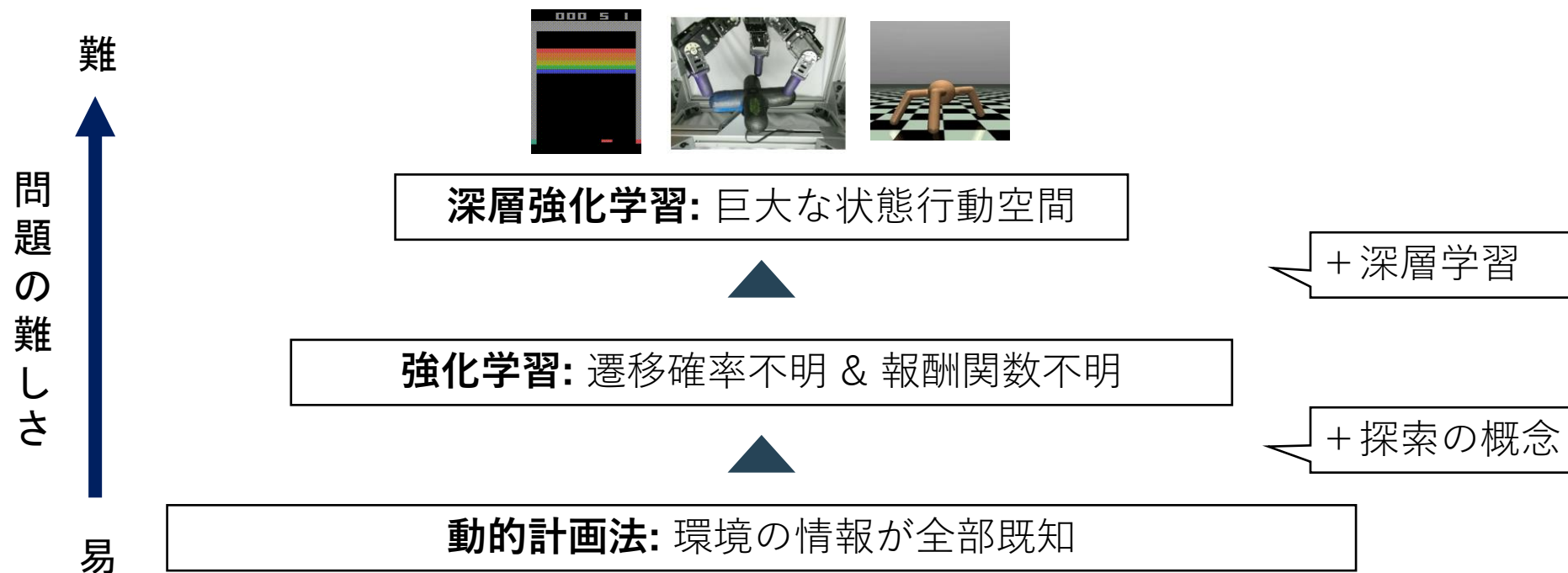


深層強化学習までの道のり

- 深層強化学習 \div 強化学習 + 深層学習
- 強化学習 \div 動的計画法 + 探索

動的計画法は深層強化学習の基礎！（深層強化学習は動的計画法のおぼけ）
動的計画法が分らないと，深層強化学習を理解するのは難しいです



この講義の目標

「**（深層）強化学習を勉強する際に便利な一般的な動的計画法**」を身に着けます。

特に，強化学習で頻出なアルゴリズム（Q学習，方策反復法，アクタークリティック…）を一般化する動的計画法について学びます。

また，第一回で省略した部分について詳しく説明します。

なぜ一般的な動的計画法を学ぶのか？：

- 深層強化学習アルゴリズムは本当に多様です。
DQN, TRPO, PPO, DDPG, TD3, AlphaGo, A3C, MCTS, …
- 全てを把握するのは大変ですが，一般化する動的計画法を学べば，それを少し変更するだけで個々のアルゴリズムの根幹を導出することができます。（勉強する内容が少なくなります！）

この講義でやらないこと

- 探索の話

探索の話は非常に難しく，そもそもバンディット問題を説明する必要があります．

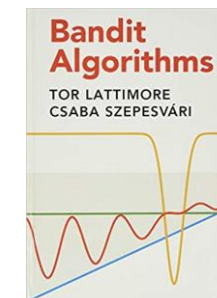
そしてバンディットと動的計画法は全く別の話です．時間の関係上今回は探索の話は省略しますが，詳しく知りたい方は次ページのBandit Algorithmsなどが参考になるかもしれません．

- 深層強化学習の話

深層強化学習では実用上導入されたトリックが多く，そのトリックが実際に効いているのかよくわかっていないものも多いです．今回はなるべく本質的な部分にだけ触れようと思うので，省略します．

教科書

- 「強化学習」 (森村哲郎)
 - 数理的な解説が中心
 - <https://www.amazon.co.jp/dp/4065155916>
- 「Bandit Algorithms」 (Csaba)
 - 探索について学びたい人におすすめ (数理)
 - <https://tor-lattimore.com/downloads/book/book.pdf>



1. 問題設定と用語を把握しよう ～テーブルマルコフ決定過程～

この章の目標

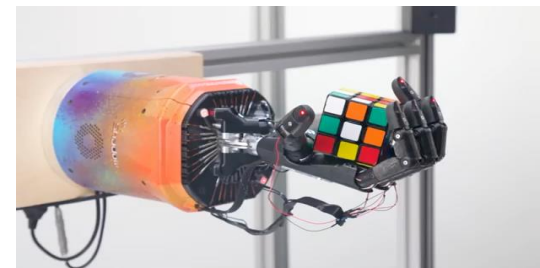
次の用語と定義は頻出です．この章では次を理解することを目標にしましょう．
まず先に定義だけ一通り導入してから，具体的な例を演習を通じて確認します．

- 逐次意思決定問題
- マルコフ決定過程の定義
- テーブルマルコフ決定過程
- 方策の定義
- プランニング問題と強化学習問題の違い
- 有限・無限マルコフ決定過程の違い
- ホライゾンとエピソード
- 期待収益・割引期待収益の定義
- 最適方策の定義

逐次意思決定問題

本講義（と他の講義）では逐次意思決定問題を解くアルゴリズムを学びます

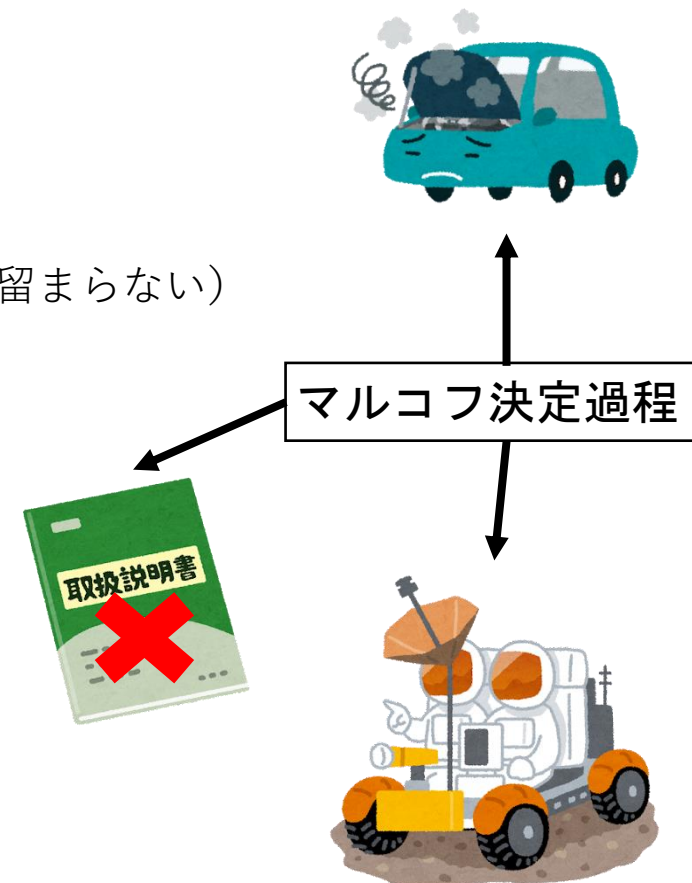
- **逐次意思決定問題**：何らかの環境で意思決定を複数回繰り返す**問題設定のこと**
 - 目標：一般には**最適な意思決定のルール**を見つけるのが目標
 - アルゴリズム：最適な意思決定のルールを**を見つけるための手順**のこと
- 逐次意思決定問題の例：
 - ロボットの制御 → 一番効率の良い動かし方は？
 - 工場の在庫管理 → 最も利益が出る在庫の調整方法は？
 - テレビゲーム → ハイスコアを出すにはどのような操作方法が求められるか？



マルコフ決定過程の意義

現実の逐次意思決定問題はとても複雑（ロボットを制御したい、だけに留まらない）

- ロボットの制御 → 一番効率の良い動かし方は？
 - ロボットの情報が無いときは動かせる？
 - ロボットを色々な環境で動かさないといけないときは？月面でも動かせる？
 - 絶対に故障しちゃいけない場合は？
 - 逆にロボットの情報も環境の情報も完璧に揃っている場合は？
 - 意思決定と結果の反映に遅延がある場合は？



全て逐次意思決定問題だが、それぞれ難易度も解き方も大きく異なる...

→ 全てに共通するアルゴリズムを考えることはほぼ不可能

→ まずは **マルコフ決定過程** という **数理モデル** を使って、シンプルな問題を考えよう！

（他の問題設定ではマルコフ決定過程を派生させればよい）

マルコフ決定過程の定義と用語

まずはマルコフ決定過程という「型」を覚えましょう。具体的な例を型に当てはめる体験は1章の最後にやります。

マルコフ決定過程 の基本的な定義

- 行動集合 A : エージェントが選択可能な行動（意思決定） ($a \in A$) の集合
- 状態集合 S : 環境の全ての状態 ($s \in S$) の集合
- 状態遷移確率 $P: S \times A \rightarrow \Delta(S)$: ある状態行動から次の状態に遷移する確率
- 報酬関数 $r: S \times A \rightarrow \mathbb{R}$: ある状態行動に対する評価
- 初期状態 $s_0 \in S$: 最初の状態
- 方策 $\pi: S \rightarrow \Delta(A)$: ある状態でのエージェントの意思決定が従う確率

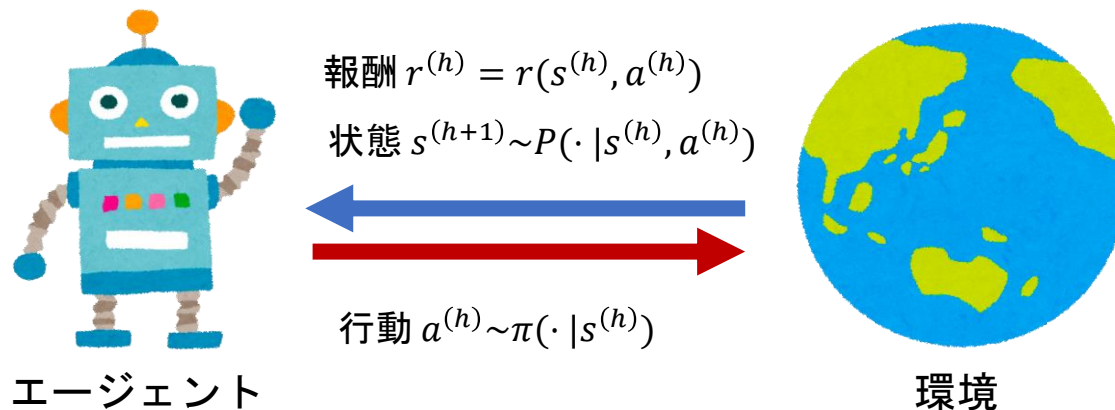
ここで、集合 S に対して $\Delta(S)$ は「 S 上の全ての確率分布の集合」を表します。また、 $f: A \rightarrow B$ の表記は、関数 f の定義域が集合 A 、終域が B であることを意味します。

マルコフ決定過程 (A, S, P, r, s_0) 上の逐次意思決定問題のうち、特に

- マルコフ決定過程の情報が全て既知の場合の意思決定問題を **プランニング問題**
- マルコフ決定過程の一部が不明（例えば P や r ）で、情報を集めながら意思決定する問題を **強化学習問題** と一般には呼称します。

文献によっては r と s_0 を確率的にするものがあります。しかし、 P の確率的な挙動によって r と s_0 の確率的な挙動を等価な形で表現することができるので、 r と s_0 は決定的でも問題ありません。

マルコフ決定過程上での逐次的な意思決定



表記： $s^{(h)}, a^{(h)}, r^{(h)}$ を、 h ステップ目の意思決定での状態，行動，発生した報酬とします。

次の手順でマルコフ決定過程は意思決定が進んでいきます：

1. 初期状態をセットします： $s^{(h)} = s_0$
2. for ステップ $h = 0, 1, 2, \dots$
 1. 状態 $s^{(h)}$ に応じて，エージェントは方策 π から行動をサンプルします： $a^{(h)} \sim \pi(\cdot | s^{(h)})$
 2. 報酬 $r^{(h)} = r(s^{(h)}, a^{(h)})$ が発生します
 3. 遷移確率 P に従って次状態に遷移します： $s^{(h+1)} \sim P(\cdot | s^{(h)}, a^{(h)})$
 4. 1に戻ります。

色々なマルコフ決定過程

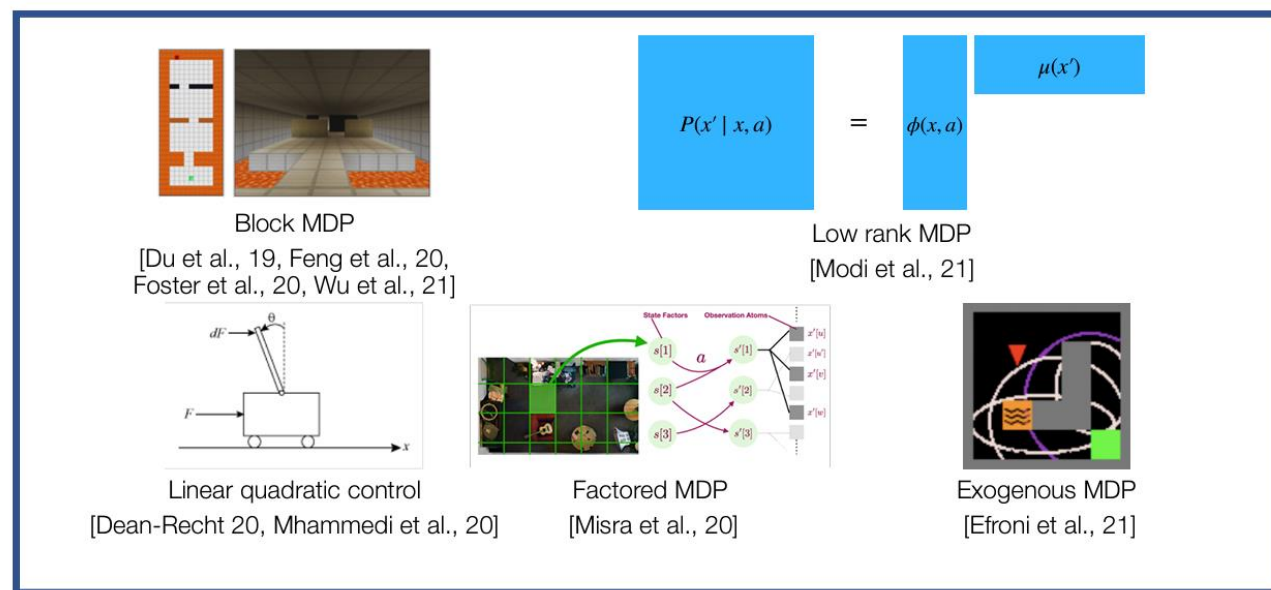
マルコフ決定過程にもいろいろあります（覚えなくていいです）。基本構造はほぼ同じです。

- **テーブルマルコフ決定過程**（一番簡単 & 勉強段階で最適！今回はこれを扱います）

- 線形マルコフ決定過程
- 線形二次レギュレータ（制御工学でよく使われます。）
- 線形混合マルコフ決定過程
- ブロックマルコフ決定過程
- Factoredマルコフ決定過程
- ベルマン完全マルコフ決定過程
- 低ランクマルコフ決定過程

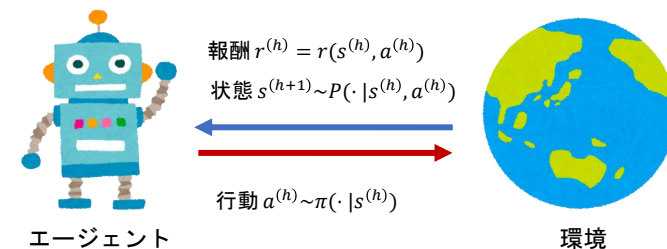
など、ほかにもたくさん

詳しくは<https://arxiv.org/abs/2209.15634>など参照



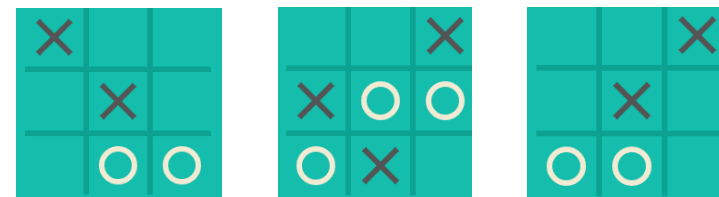
https://rltheorybook.github.io/colt21_part3.pdf

テーブルマルコフ決定過程



「テーブルマルコフ決定過程」：行動集合 A と状態集合 S が有限なマルコフ決定過程

- 例：三目並べ
 - $A = \{\text{左上に置く, 中上に置く, ..., 中下に置く, 右下に置く}\} \rightarrow$ 全部で9個
 - $S = \{\text{○と×のパターン全て}\} \rightarrow$ 最大でも $3^9 \leq 20000$ 通り
 - 遷移確率 P ：敵次第.
 - 報酬関数 r ：勝利すると+1など. 設計次第.
 - 初期状態 s_0 ：何も無い盤面



どんなときにテーブルマルコフ決定過程を考えるのか？

- 三目並べのような単純な問題
- 強化学習研究のアイデア出し
 - 「テーブルマルコフ決定過程でアイデア出し→深層強化学習に反映」が歴史的に繰り返されてきました
 - 逆に、テーブルでうまくいかないアイデアは深層でもうまくいかないことが多いです.

テーブルマルコフ決定過程と行列

「テーブルマルコフ決定過程」：行動集合 A と状態集合 S が有限なマルコフ決定過程

行動集合と状態集合が有限ならば、遷移確率 P 、報酬関数 r 、そして方策 π は**行列として表現できます**。

(行列形式にすると後で便利です)

試しに $A = \{a_0, a_1\}$, $S = \{s_0, s_1, s_2\}$ のテーブルマルコフ決定過程を考えましょう。

つまり、 $|A| = 2$, $|S| = 3$ のテーブルマルコフ決定過程です。このとき、次のような行列で P と r と π は表現できます

$P =$ 行動 × 状態
($S \times A$) × S の行列

状態		
$P(s_0 s_0, a_0)$	$P(s_1 s_0, a_0)$	$P(s_2 s_0, a_0)$
$P(s_0 s_0, a_1)$	$P(s_1 s_0, a_1)$	$P(s_2 s_0, a_1)$
$P(s_0 s_1, a_0)$	$P(s_1 s_1, a_0)$	$P(s_2 s_1, a_0)$
$P(s_0 s_1, a_1)$	$P(s_1 s_1, a_1)$	$P(s_2 s_1, a_1)$
$P(s_0 s_2, a_0)$	$P(s_1 s_2, a_0)$	$P(s_2 s_2, a_0)$
$P(s_0 s_2, a_1)$	$P(s_1 s_2, a_1)$	$P(s_2 s_2, a_1)$

$r =$ 状態
 $S \times A$ の行列

行動	
$r(s_0, a_0)$	$r(s_0, a_1)$
$r(s_1, a_0)$	$r(s_1, a_1)$
$r(s_2, a_0)$	$r(s_2, a_1)$

$\pi =$ 状態
 $S \times A$ の行列

行動	
$\pi(a_0 s_0)$	$\pi(a_1 s_0)$
$\pi(a_0 s_1)$	$\pi(a_1 s_1)$
$\pi(a_0 s_2)$	$\pi(a_1 s_2)$

テーブルマルコフ決定過程を作ってみよう！

Numpyを使って、テーブルマルコフ決定過程を定義通りに適当に作ってみましょう。

行動数 2, 状態数 3 のマルコフ決定過程を試しに作ってみます。 (具体的な例は1章の最後に実装します)

- 行動集合 A : $A = \{a_0, a_1\}$ とします。
- 状態集合 S : $S = \{s_0, s_1, s_2\}$ とします。

```
A, S = 2, 3 # 行動, 状態集合のサイズ
A_set = np.arange(A) # 行動集合
S_set = np.arange(S) # 状態集合
```

続いて、状態遷移確率と報酬関数を作ってみましょう。

S, A が特に有限集合であれば、前ページのように行列として扱うことができます。

`np.random.rand` を使って適当に作ります。

- 状態遷移確率 $P \in \mathbb{R}^{(S \times A) \times S}$
- 報酬関数 $r \in \mathbb{R}^{S \times A}$

```
P = np.random.rand(S*A, S)
# 正規化して確率にします
P = P / np.sum(P, axis=1, keepdims=True)
P = P.reshape(S, A, S) # 遷移確率
rew = np.random.rand(S, A) # 報酬関数
```

マルコフ決定過程で逐次意思決定を体験しよう！

前ページで作った (A, S, P, r, s_0) は立派なマルコフ決定過程です。例えば

```
print("行動a1, 状態s2での次状態が出る確率:", P[1, 2])
print("行動a0, 状態s1での報酬の値:", rew[0, 1])
```

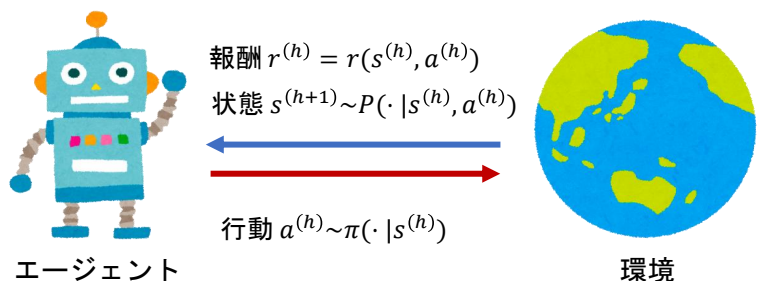
のようにして、各状態行動での遷移確率や報酬の値にアクセスすることができます。
続いて、方策を作ってみましょう。方策も行列として表現できます。

```
policy = np.random.rand(S, A)
# 正規化して確率にします
policy = policy / np.sum(policy, axis=1, keepdims=True) # 方策

# 試しに各状態での行動の確率を確認してみましょう
print("状態s0での行動の確率分布:", policy[0])
print("状態s1での行動の確率分布:", policy[1])
print("状態s2での行動の確率分布:", policy[2])
```

マルコフ決定過程で逐次意思決定を体験しよう！

作ったマルコフ決定過程と方策で，逐次意思決定してみましょう．



```
s = 0 # 初期状態です
a = np.random.choice(A, p=policy[s]) # 状態sで，方策 $\pi(\cdot|s)$ から行動をサンプルします
# 状態s，行動aで，遷移確率 $P(\cdot|s, a)$ から次状態をサンプルします
next_s = np.random.choice(S, p=P[s, a])
reward = rew[s, a] # 報酬を得ます
# 同じことを繰り返します
s = next_s # 次状態を現在の状態とします
a = np.random.choice(A, p=policy[s])
next_s = np.random.choice(S, p=P[s, a])
reward = rew[s, a]
s = next_s
# ...
```


マルコフ決定過程で何がしたいの？

前ページで実行したコードは、現実的な問題をかなり単純化したものとみなせます。
例えば以下のように、在庫管理問題をモデル化することも可能です。

$A = \{a_0 = \text{在庫処分}, a_1 = \text{商品を発注}\}$, $S = \{s_0 = \text{在庫少ない}, s_1 = \text{在庫普通}, s_2 = \text{在庫多め}\}$
 $r(a_0, s_0) = -100$ # 在庫少ないのに在庫処分すると-100万円のコスト
 $r(a_1, s_0) = 30$ # 在庫少ないときに商品を発注すると+30万円の利益
在庫少ないときに商品を発注すると、在庫が少し増える
 $P(s_0 | s_0, a_1) = 0.1, P(s_1 | s_0, a_1) = 0.7, P(s_2 | s_0, a_1) = 0.2$

(チャレンジ：他の報酬 $r(a_0, s_1)$ や遷移確率 $P(s_0 | s_1, a_0)$ なども設計してみよう)

上のような在庫管理問題を考えると、
長期的な利益（報酬）の総和を最大化させるような方策（最適方策）が欲しくなります。
(その場しのぎで在庫管理をすると、後々困りますよね)

「最適方策を求める」 \equiv 「マルコフ決定過程での目標」です。

有限・無限マルコフ決定過程

最適方策：長期的な報酬の総和を最大化させるような方策
考慮する意思決定の長さが**有限 or 無限**で問題設定（マルコフ決定過程の種類）が変わります

有限マルコフ決定過程：意思決定の長さが有限

- 10秒だけ稼働させるロボット
- 制限時間が決められたテレビゲーム



無限マルコフ決定過程：意思決定の長さが無限

- 永遠に稼働させる工場
- マインクラフト
- 将棋（ほぼ無限）



- ちなみに、意思決定の長さが1の強化学習問題のことを、**バンディット問題**と呼びます。
- 深層強化学習では無限を想定して設計されたアルゴリズムが多いです。DQN, SAC, TD3, DDPG, などは無限が多め（無限用のアルゴリズムでも、実用上は工夫すれば有限用に調整できることがあります）。
- 一方、モンテカルロ法で方策評価する場合は有限が想定されることが多いです。PPO, TRPOなどは有限が多め。

有限マルコフ決定過程での期待収益と最適方策

期待収益はマルコフ決定過程で最大化させる**長期的な報酬の総和の期待値**のことであり、最適方策は**期待収益を最大化させる方策**のことです。それぞれちゃんとした定義を見てみましょう。

有限マルコフ決定過程の期待収益と最適方策

- 意思決定が始まってから終わるまでの1巡は**エピソード**と呼びます。
- エピソードの長さを**ホライゾン**と呼びます。 H をホライゾンとします。
- 方策 π が1エピソードで出会う報酬を $r^{(0)}, r^{(1)}, r^{(2)}, \dots, r^{(H-1)}$ とします。
このとき、そのエピソードでの**収益**は $R = \sum_{h=0}^{H-1} r^{(h)}$ で定義されます。
- **期待収益**： R の期待値のこと。つまり、 $\mathbb{E}^{\pi}[R]$ のこと。
- **最適方策**： $\mathbb{E}^{\pi}[R]$ を最大にする方策のこと。つまり、 $\pi^* = \operatorname{argmax}_{\pi} \mathbb{E}^{\pi}[R]$ のこと。

有限マルコフ決定過程を体験しよう！

有限マルコフ決定過程（100ステップだけ意思決定します）

```
s = 0 # 初期状態です
H = 10 # ホライゾンです
R = 0 # 収益をカウントします
for _ in range(H): # 有限マルコフ決定過程です.
    a = np.random.choice(A, p=policy[s])
    next_s = np.random.choice(S, p=P[s, a])
    reward = rew[s, a]
    R += reward
    print(f"状態と行動: s={s}, a={a}")
    s = next_s
print("このエピソードの収益: ", R)
```

補足：有限マルコフ決定過程で期待収益を近似してみよう

- (復習) 期待収益： R の期待値のこと。つまり、 $\mathbb{E}^\pi[R]$ のこと。
- (シミュレータなどで) 同じ環境を N 個用意できる場合、集計のサンプル R_0, R_1, \dots, R_{N-1} を生成して、

$\mathbb{E}^\pi[R] \approx \frac{1}{N} \sum_{n=0}^{N-1} R_n$ として期待収益を近似できます。

- このような近似方法を**モンテカルロ法**と言います。 N が大きいほど精度が上がります。

```
H = 10 # ホライゾンです
R_sum = 0 # 期待収益の計算用です
N = 1000 # モンテカルロ法で使用するサンプルの数です。

for _ in range(N):
    s = 0 # 初期状態です
    R = 0 # 収益をカウントします
    for _ in range(H):
        a = np.random.choice(A, p=policy[s])
        next_s = np.random.choice(S, p=P[s, a])
        reward = rew[s, a]
        R += reward
        s = next_s
    R_sum += R
print("モンテカルロ法された期待収益:", R_sum / N)
```

無限マルコフ決定過程での期待収益

無限マルコフ決定過程では、有限と同じような収益の定義を使うと困ったことになります。

無限マルコフ決定過程の期待収益の困りポイント

- 無限マルコフ決定過程ではホライゾン $H = \infty$ です。
- このとき、収益は $R = r^{(0)} + r^{(1)} + \dots$ ですが、収益の値も ∞ を取る可能性があります。
例えば方策Aと方策Bのどちらも ∞ の収益を取る場合、AとBのどちらがすぐれた方策かわかりません。
(適当な在庫管理も優れた在庫管理もどちらの収益も ∞ になってしまいます。)
- つまり、このままでは**適当な方策を考えても最適方策になってしまう**かもしれません。

そこで、無限マルコフ決定過程では期待収益を**割引率**($0 < \gamma < 1$)を使って無限にならないように調整します。

- 割引収益** : $R = r^{(0)} + \gamma r^{(1)} + \gamma^2 r^{(2)} + \dots = \sum_{h=0}^{\infty} \gamma r^{(h)}$
- 期待割引収益** : $\mathbb{E}^{\pi}[R]$ のこと。 $0 < \gamma < 1$ なので、 $\mathbb{E}^{\pi}[R] \leq \frac{\text{報酬の最大値}}{1-\gamma}$ であり、有限な値。

無限マルコフ決定過程を体験しよう！

無限マルコフ決定過程

```
s = 0 # 初期状態です
R = 0 # 割引収益をカウントします
gamma = 0.7 # 割引率です
t = 0 # 経過したステップ数です
while True: # 無限マルコフ決定過程です.
    a = np.random.choice(A, p=policy[s])
    next_s = np.random.choice(S, p=P[s, a])
    reward = rew[s, a]
    R += gamma ** t * reward
    s = next_s
    print(f"{t} ステップ目までの割引収益 : ", R)
    t += 1
```

補足：無限マルコフ決定過程で期待収益を近似してみよう

- 無限マルコフ決定過程では、エピソードという概念が本来はありません ($H = \infty$ なので)
- しかし、割引率のおかげで疑似的なホライゾンが設定されているとみなすことができます。

例えば $\gamma = 0.7$ ならば $\gamma^{30} \approx 10^{-5}$ なので、30ステップ過ぎてから、 $R = \sum_{h=0}^{\infty} \gamma^h r^{(h)} \approx \sum_{h=0}^{30} \gamma^h r^{(h)}$ と近似しても問題なさそうです。（どれくらいのホライゾンで近似するかは割引率の値によって異なります。考えてみましょう。）

```
gamma = 0.7 # 割引率です
H = 30 # 十分な精度が期待できるホライゾンです
R_sum = 0 # 期待収益の計算用です
N = 100 # モンテカルロ法で使用するサンプルの数です.
for _ in range(N):
    s = 0 # 初期状態です
    R = 0 # 収益をカウントします
    for _ in range(H):
        a = np.random.choice(A, p=policy[s])
        next_s = np.random.choice(S, p=P[s, a])
        reward = rew[s, a]
        R += gamma ** t * reward
        s = next_s
    R_sum += R
print("モンテカルロ法された割引期待収益:", R_sum / N)
```


逐次意思決定問題のまとめと具体例（有限ホライゾン）

チャレンジ！次のじゃんけんゲームをNumpyで実装して体験しましょう

自明な問題（ $H = 1$ & 環境既知）

プレイ回数：1回

ルール：じゃんけんに勝つと+1，それ以外は+0の報酬が発生します。

情報：A君は(グー, チョキ, パー) = (0.2, 0.5, 0.3)の確率で出してくれます。

ゴール：最も報酬が高くなる手を出してください



0.2 0.5 0.3

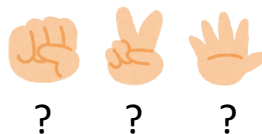
バンディット問題（ $H = 1$ & 環境未知）

プレイ回数：M回

ルール：じゃんけんに勝つと+1，それ以外は+0の報酬が発生します。

情報：A君は(グー, チョキ, パー) = (?, ?, ?)の確率で出してくれます。

ゴール：Mゲーム以内に，最も報酬が高くなる手を見つけてください



? ? ?

	ホライゾン	環境の情報
バンディット問題	$H = 1$	未知
プランニング問題	$H \geq 1$	既知
強化学習問題	$H \geq 1$	未知

有限ホライゾンプランニング問題（ $H \geq 1$ & 環境既知）

プレイ回数：1回

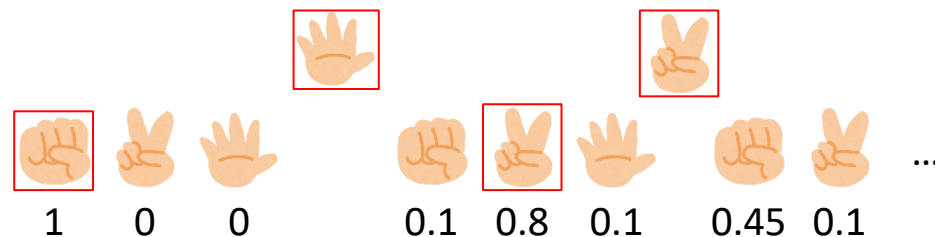
ルール： H 回じゃんけんをして， H 回目にパーで勝つときだけ+1，それ以外は+0の報酬が発生します。

情報：A君は最初必ずグーを出します。次以降，A君は直前に出された手以外を高い確率で出すとします。

例えば，(あなた, A君) = (グー, チョキ)なら，次は(0.1, 0.1, 0.8)。また，例えば

(あなた, A君) = (グー, グー)なら，次は(0.1, 0.45, 0.45)で手を出します。

ゴール：最も勝率が高くなる方策を出してください



1 0 0 0.1 0.8 0.1 0.45 0.1 ...

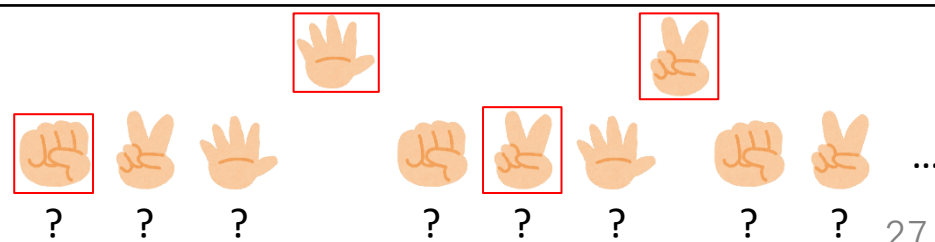
有限ホライゾン強化学習問題（ $H \geq 1$ & 環境未知）

プレイ回数：M回

ルール： H 回じゃんけんをして， H 回目にパーで勝つときだけ+1，それ以外は+0の報酬が発生します。

情報：A君がどのような条件で何を出すか，一切不明です。

ゴール：Mゲーム以内に，最も勝率が高くなる方策を出してください



? ? ? ? ? ? ? ? 27

逐次意思決定問題のまとめと具体例（無限ホライゾン）

チャレンジ！次のじゃんけんゲームをNumpyで実装して体験しましょう

自明な問題（ $H = 1$ & 環境既知）

プレイ回数：1回

ルール：普通のじゃんけんをします。

情報：A君は(グー, チョキ, パー) = (0.2, 0.5, 0.3)の確率で出してくれます。

ゴール：最も勝率が高くなる手を出してください



0.2 0.5 0.3

バンディット問題（ $H = 1$ & 環境未知）

プレイ回数：M回

ルール：普通のじゃんけんをします。

情報：A君は(グー, チョキ, パー) = (?, ?, ?)の確率で出してくれます。

ゴール：Mゲーム以内に、最も勝率が高くなる手を見つけてください



? ? ?

	ホライゾン	環境の情報
バンディット問題	$H = 1$	未知
プランニング問題	$H \geq 1$	既知
強化学習問題	$H \geq 1$	未知

無限ホライゾンプランニング問題（ $H = \infty$ & 環境既知）

プレイ回数：1回

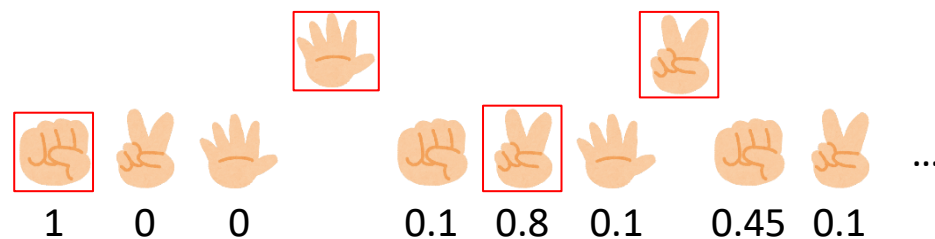
ルール：無限回じゃんけんをします。パーで勝つと+1, それ以外は0の報酬が発生します。

情報：A君は最初必ずグーを出します。次以降、A君は直前に出された手以外を高い確率で出すとします。

例えば、(あなた, A君) = (グー, チョキ)なら、次は(0.1, 0.1, 0.8)。また、例えば

(あなた, A君) = (グー, グー)なら、次は(0.1, 0.45, 0.45)で手を出します。

ゴール：割引累積報酬和が最も高くなる方策を出してください



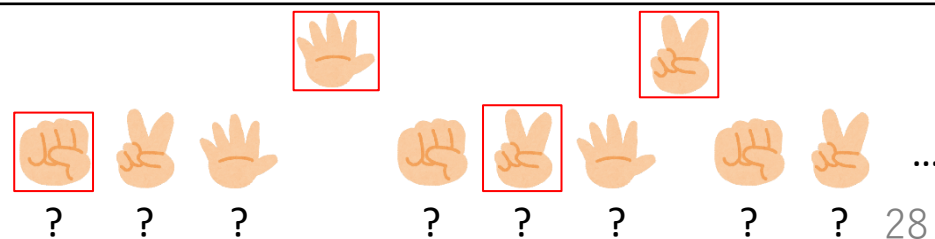
無限ホライゾン強化学習問題（ $H = \infty$ & 環境未知）

プレイ回数：1回（無限ホライゾンなので）

ルール：無限回じゃんけんをします。パーで勝つと+1, それ以外は0の報酬が発生します。

情報：A君がどのような条件で何を出すか、一切不明です。

ゴール：割引累積報酬和が最も高くなる方策を、なるべく少ないじゃんけん数で出してください。



2. プランニング問題での最適方策を計算しよう ～動的計画法～

この章の目標

- 「最適方策を求める」は、ほぼ全ての逐次意思決定問題で共通の目標です。
- この章では、テーブルマルコフ決定過程でのプランニング問題**（定義を思い出してください）における**最適方策の求め方を学びます**。
- つまり、 P と r が与えられたときに π^* を求めます。
- また、ここから先は（時間と表記の都合上）**無限マルコフ決定過程についてのみ扱います**。
有限マルコフ決定過程については各自調べてみてください。

$P =$ $(S \times A) \times S$ の行列		状態			$r =$ 状態 $S \times A$ の行列
		$P(s_0 s_0, a_0)$	$P(s_1 s_0, a_0)$	$P(s_2 s_0, a_0)$	
		$P(s_0 s_0, a_1)$	$P(s_1 s_0, a_1)$	$P(s_2 s_0, a_1)$	
		$P(s_0 s_1, a_0)$	$P(s_1 s_1, a_0)$	$P(s_2 s_1, a_0)$	
		$P(s_0 s_1, a_1)$	$P(s_1 s_1, a_1)$	$P(s_2 s_1, a_1)$	
		$P(s_0 s_2, a_0)$	$P(s_1 s_2, a_0)$	$P(s_2 s_2, a_0)$	
		$P(s_0 s_2, a_1)$	$P(s_1 s_2, a_1)$	$P(s_2 s_2, a_1)$	

行動	
$r(s_0, a_0)$	$r(s_0, a_1)$
$r(s_1, a_0)$	$r(s_1, a_1)$
$r(s_2, a_0)$	$r(s_2, a_1)$



行動	
$\pi(a_0 s_0)$	$\pi(a_1 s_0)$
$\pi(a_0 s_1)$	$\pi(a_1 s_1)$
$\pi(a_0 s_2)$	$\pi(a_1 s_2)$

最適方策の求め方

テーブルマルコフ決定過程でのプランニング問題における π^* の求め方は大きく 2 種類あります.

1. **動的計画法** : DQNやActor-critic系のアルゴリズムは動的計画法に基づいています.
2. **線形計画法** : 発展的なアルゴリズムを導出する際は非常に重要ですが, 勉強の序盤で目にすることはほぼないと思います.

(オフポリシー方策評価系のアルゴリズムで重要になります. 詳しくは<https://arxiv.org/abs/2001.01866> など)

動的計画法とは**マルコフ決定過程の帰納的な性質を利用して最適方策を求めるアルゴリズムの総称**です. 具体的なアルゴリズムとして,

- **価値反復法**
- **方策反復法**

の 2 つを学びます.

まずは「帰納的な性質」について確認してみましょう.

無限マルコフ決定過程での価値関数

動的計画法は、次の**価値関数**を利用して最適方策を見つけに行きます。

- 割引収益（復習です） : $R = r^{(0)} + \gamma r^{(1)} + \gamma^2 r^{(2)} + \dots = \sum_{h=0}^{\infty} \gamma^h r^{(h)}$
- π の**状態価値関数** $V^\pi \in \mathbb{R}^S : V^\pi(s) = E^\pi[R \mid s^{(0)} = s]$
つまり $V^\pi(s)$ は「状態 s から方策 π で無限ステップ逐次意思決定したとき、期待される割引収益」です。
- π の**状態行動価値関数** $Q^\pi \in \mathbb{R}^{S \times A} : Q^\pi(s, a) = E^\pi[R \mid s^{(0)} = s, a^{(0)} = a]$
つまり $Q^\pi(s, a)$ は「状態 s 、行動 a から方策 π で無限ステップ逐次意思決定したとき、期待される割引収益」です。

それぞれV関数、Q関数とも呼ばれます。定義から、 $V^\pi(s) = \sum_{a \in A} \pi(a \mid s) Q^\pi(s, a)$ であることに注意しましょう。

$A = \{a_0, a_1\}$, $S = \{s_0, s_1, s_2\}$ のマルコフ決定過程を考えると、次のように行列で表現することもできます。

		行動	
$V^\pi =$ <div>Sの行列</div>	状態		
	$V^\pi(s_0)$	$Q^\pi(s_0, a_0)$	$Q^\pi(s_0, a_1)$
	$V^\pi(s_1)$	$Q^\pi(s_1, a_0)$	$Q^\pi(s_1, a_1)$
	$V^\pi(s_2)$	$Q^\pi(s_2, a_0)$	$Q^\pi(s_2, a_1)$

便利な略記

さらに天下りのですが、適当な行列 $V \in \mathbb{R}^S$ と $Q \in \mathbb{R}^{S \times A}$ について、次の略記を導入します。

- 方策 π について、 $\langle \pi, Q \rangle \in \mathbb{R}^S$ は $\langle \pi, Q \rangle(s) = \sum_{a \in A} \pi(a | s) Q(s, a)$ を満たす行列とする。
- $\max_a Q \in \mathbb{R}^S$ は $(\max_a Q)(s) = \max_a Q(s, a)$ を満たす行列とする。
- 遷移 P について、 $PV \in \mathbb{R}^{S \times A}$ は $(PV)(s, a) = \sum_{s' \in S} P(s' | s, a) V(s')$ を満たす行列とする。

これを使うと、 $V^\pi(s) = \sum_{a \in A} \pi(a | s) Q^\pi(s, a)$ のことを $V^\pi = \pi Q^\pi$ と簡単に書くことができます。
(発展的な論文などでたまに出きます)

numpyでは次の処理と同じです。

```
# QとVを適当に作ります
Q = np.random.rand(S, A)
V = np.random.rand(S)
policy_Q = (policy * Q).sum(axis=-1) # <math>\langle \pi, Q \rangle</math>と同じです
max_Q = Q.max(axis=-1) #  $\max_a Q$ と同じです
PV = P @ V #  $PV$ と同じです
```

(@を使うと、 $P.reshape(-1, S) * V.reshape(1, S)).sum(axis=-1)$ を簡単に書くことができます。調べてみてください)

無限マルコフ決定過程でのベルマン方程式

動的計画法では、次のベルマン方程式を利用して最適方策を見つけに行きます。

無限マルコフ決定過程では次の**期待ベルマン方程式**が成立します。

$$Q^\pi = r + \gamma P \langle \pi, Q^\pi \rangle$$

さらに、最適方策 π^* については次の**最適ベルマン方程式**が成立します。

$$Q^{\pi^*} = r + \gamma P \left(\max_a Q^{\pi^*} \right)$$

証明は省略します。



などを参考にしてください。

無限マルコフ決定過程での期待ベルマン方程式（行列表現）

$$\begin{array}{c} \text{行動} \times \text{状態} \\ \begin{array}{|c|} \hline Q^\pi(s_0, a_0) \\ \hline Q^\pi(s_1, a_0) \\ \hline Q^\pi(s_2, a_0) \\ \hline Q^\pi(s_0, a_1) \\ \hline Q^\pi(s_1, a_1) \\ \hline Q^\pi(s_2, a_1) \\ \hline \end{array} \\ Q^\pi \\ S \times A \text{の行列} \end{array} = \begin{array}{c} \begin{array}{|c|} \hline r(s_0, a_0) \\ \hline r(s_1, a_0) \\ \hline r(s_2, a_0) \\ \hline r(s_0, a_1) \\ \hline r(s_1, a_1) \\ \hline r(s_2, a_1) \\ \hline \end{array} \\ r \\ S \times A \text{の行列} \end{array} + \gamma \cdot \begin{array}{c} \text{状態} \\ \begin{array}{|c|c|c|} \hline P(s_0 | s_0, a_0) & P(s_1 | s_0, a_0) & P(s_2 | s_0, a_0) \\ \hline P(s_0 | s_0, a_1) & P(s_1 | s_0, a_1) & P(s_2 | s_0, a_1) \\ \hline P(s_0 | s_1, a_0) & P(s_1 | s_1, a_0) & P(s_2 | s_1, a_0) \\ \hline P(s_0 | s_1, a_1) & P(s_1 | s_1, a_1) & P(s_2 | s_1, a_1) \\ \hline P(s_0 | s_2, a_0) & P(s_1 | s_2, a_0) & P(s_2 | s_2, a_0) \\ \hline P(s_0 | s_2, a_1) & P(s_1 | s_2, a_1) & P(s_2 | s_2, a_1) \\ \hline \end{array} \\ P \\ (S \times A) \times S \\ \text{の行列} \end{array} \cdot \begin{array}{c} \begin{array}{|c|} \hline \langle \pi, Q^\pi \rangle(s_0) \\ \hline \langle \pi, Q^\pi \rangle(s_1) \\ \hline \langle \pi, Q^\pi \rangle(s_2) \\ \hline \end{array} \\ \langle \pi, Q^\pi \rangle \\ S \text{の行列} \end{array}$$

無限マルコフ決定過程での最適ベルマン方程式（行列表現）

状態

$Q^{\pi^*}(s_0, a_0)$
$Q^{\pi^*}(s_1, a_0)$
$Q^{\pi^*}(s_2, a_0)$
$Q^{\pi^*}(s_0, a_1)$
$Q^{\pi^*}(s_1, a_1)$
$Q^{\pi^*}(s_2, a_1)$

=

$r(s_0, a_0)$
$r(s_1, a_0)$
$r(s_2, a_0)$
$r(s_0, a_1)$
$r(s_1, a_1)$
$r(s_2, a_1)$

+ $\gamma \cdot$

$P(s_0 s_0, a_0)$	$P(s_1 s_0, a_0)$	$P(s_2 s_0, a_0)$
$P(s_0 s_0, a_1)$	$P(s_1 s_0, a_1)$	$P(s_2 s_0, a_1)$
$P(s_0 s_1, a_0)$	$P(s_1 s_1, a_0)$	$P(s_2 s_1, a_0)$
$P(s_0 s_1, a_1)$	$P(s_1 s_1, a_1)$	$P(s_2 s_1, a_1)$
$P(s_0 s_2, a_0)$	$P(s_1 s_2, a_0)$	$P(s_2 s_2, a_0)$
$P(s_0 s_2, a_1)$	$P(s_1 s_2, a_1)$	$P(s_2 s_2, a_1)$

•

$(\max_a Q^{\pi^*})(s_0)$
$(\max_a Q^{\pi^*})(s_1)$
$(\max_a Q^{\pi^*})(s_2)$

Q^{π^*}

$S \times A$ の行列

r

$S \times A$ の行列

P

$(S \times A) \times S$
の行列

$\max_a Q^{\pi^*}$

S の行列

なぜベルマン方程式を考えるの？

ここまで、 $Q^\pi = r + \gamma P\langle \pi, Q^\pi \rangle$ と $Q^{\pi^*} = r + \gamma P\left(\max_a Q^{\pi^*}\right)$ という方程式を導入しました。

なぜこんな方程式を導入したのでしょうか？実は、 $Q \in \mathbb{R}^{S \times A}$ について、次が成立します：

$$Q = r + \gamma P\langle \pi, Q \rangle \Leftrightarrow Q = Q^\pi$$

$$Q = r + \gamma P\left(\max_a Q\right) \Leftrightarrow Q = Q^{\pi^*}$$

つまり、**期待（最適）ベルマン方程式を満たす Q は Q^π （ Q^{π^*} ）ただ一つです！**

動的計画法はこの性質を利用して Q^π or Q^{π^*} を見つけ、見つけた Q^π or Q^{π^*} を使って最適方策を見つけに行きます。

証明は省略します。



などを参考にしてください。

無限マルコフ決定過程でのベルマン作用素

Q^π と Q^{π^*} を見つけるときに便利なのが次の**ベルマン作用素**です. (これは方程式ではなく関数です!)

- **期待ベルマン作用素** $B^\pi: \mathbb{R}^{S \times A} \rightarrow \mathbb{R}^{S \times A}$ は次を満たす**関数のこと**: $B^\pi(Q) = r + \gamma P\langle \pi, Q \rangle$
- **最適ベルマン作用素** $B: \mathbb{R}^{S \times A} \rightarrow \mathbb{R}^{S \times A}$ は次を満たす**関数のこと**: $B(Q) = r + \gamma P\left(\max_a Q\right)$

適当な $Q_0 \in \mathbb{R}^{S \times A}$ に対してベルマン作用素を繰り返し適用すると, 次が成立します.

- $Q_{k+1} = B^\pi(Q_k)$ によって $Q_0, Q_1, \dots, Q_k, Q_{k+1}, \dots$ と更新し続けると, $\lim_{k \rightarrow \infty} Q_k = Q^\pi$ に収束します.
- $Q_{k+1} = B(Q_k)$ によって $Q_0, Q_1, \dots, Q_k, Q_{k+1}, \dots$ と更新し続けると, $\lim_{k \rightarrow \infty} Q_k = Q^{\pi^*}$ に収束します.

つまり, **ベルマン作用素を収束するまで繰り返し適用すると, Q^π と Q^{π^*} を発見できます.**

収束していない場合は $\|Q_k - B^\pi(Q_k)\|_\infty$ が正の値を取ります. この値を**ベルマン誤差**と呼びます.

証明は省略します。



などを参考にしてください。

ベルマン期待作用素を実装しよう！

- （復習）期待ベルマン作用素 $B^\pi: \mathbb{R}^{S \times A} \rightarrow \mathbb{R}^{S \times A}$ は次を満たす関数のこと： $B^\pi(Q) = r + \gamma P \langle \pi, Q \rangle$

```
# ベルマン期待作用素です
def Bellman_expected_operator(rew, P, gamma, Q, policy):
    S, A = Q.shape

    # 各入力のshapeが正しいか確認します
    assert Q.shape == (S, A)
    assert policy.shape == (S, A)
    assert rew.shape == (S, A)
    assert P.shape == (S, A, S)

    policy_Q = (policy * Q).sum(axis=1) # <πQ>です
    P_policy_Q = P @ policy_Q # P<πQ>です
    B_policy_Q = rew + gamma * P_policy_Q # ベルマン期待作用素の結果です
    return B_policy_Q
```

ベルマン最適作用素を実装しよう！

- （復習）最適ベルマン作用素 $B : \mathbb{R}^{S \times A} \rightarrow \mathbb{R}^{S \times A}$ は次を満たす関数のこと： $B(Q) = r + \gamma P \left(\max_a Q \right)$

```
# ベルマン最適作用素です
def Bellman_optimal_operator(rew, P, gamma, Q):
    S, A = Q.shape

    # 各入力のshapeが正しいか確認します
    assert Q.shape == (S, A)
    assert rew.shape == (S, A)
    assert P.shape == (S, A, S)

    max_Q = Q.max(axis=1) # maxQです
    P_max_Q = P @ max_Q # P(maxQ)です
    BQ = rew + gamma * P_max_Q # ベルマン最適作用素の結果です
    return BQ
```

ベルマン作用素の収束を観察しよう！

(復習) $Q_{k+1} = \mathcal{B}^\pi(Q_k)$ によって $Q_0, Q_1, \dots, Q_k, Q_{k+1}, \dots$ と更新し続けると, $\lim_{k \rightarrow \infty} Q_k = Q^\pi$ に収束します.

```
Q = np.random.rand(S, A) # 適当なQの初期値を生成します
for _ in range(100): # 無限回更新するのは大変なので, 適当な大きい回数で打ち切ります
    Q = Bellman_expected_operator(rew, P, gamma, Q, policy)

# チャレンジ! ここで得られたQとモンテカルロ法した期待収益 $\mathbb{E}^\pi[R]$ を比較してみましょう.
```

(復習) $Q_{k+1} = \mathcal{B}(Q_k)$ によって $Q_0, Q_1, \dots, Q_k, Q_{k+1}, \dots$ と更新し続けると, $\lim_{k \rightarrow \infty} Q_k = Q^{\pi^*}$ に収束します.

```
Q = np.random.rand(S, A) # 適当なQの初期値を生成します
For _ in range(100): # 無限回更新するのは大変なので, 適当な大きい回数で打ち切ります
    Q = Bellman_optimal_operator(rew, P, gamma, Q)

# チャレンジ! ここで得られたQと, Bellman_expected_operatorで得られたQを比較してみましょう.
```

無限マルコフ決定過程での価値反復法


価値反復法は、最適ベルマン作用素によって Q^{π^*} を見つけ、見つけた Q^{π^*} を使って π^* を見つけるアルゴリズムです。

- 入力：報酬行列 r , 遷移確率行列 P , 割引率 γ , イテレーションの回数 K
 - 出力：方策 π
1. Q_0 を要素が全て0の $S \times A$ の行列で初期化する
 2. for $k = 0, 1, 2, \dots, K$
 3. $Q_{k+1} = \mathcal{B}(Q_k)$
 4. Q_K の**貪欲方策**を出力する。

$Q \in \mathbb{R}^{S \times A}$ の貪欲方策は、 $Q(s, a)$ が最も高い行動を常に選択する方策のことです。

つまり、全ての状態についての次を満たす方策 π_{greedy} です： $\pi_{\text{greedy}} = \arg \max_{\pi} \langle \pi, Q \rangle$

簡便化のため、 $\arg \max_{\pi}$ は「すべての状態で $\sum_{a \in A} \pi(a | s) Q(s, a)$ を最大化する方策」として使っています。以降もこの表記を使用します。

アルゴリズムの精度の証明は省略します。  などを参考にしてください。

価値反復法を実装しよう！

```
def compute_greedy_policy(Q):  
    # 貪欲方策を計算する関数です  
    policy = np.zeros((S, A))  
    policy[np.arange(S), Q.argmax(axis=1)] = 1.0  
    return policy
```

```
def ValueIteration(rew, P, gamma, iteration):  
    S, A = rew.shape  
    Q = np.zeros((S, A)) # 適当なQの初期値を生成します  
    for _ in range(iteration): # ベルマン最適作用素を繰り返し適用します  
        Q = Bellman_optimal_operator(rew, P, gamma, Q)  
    policy = compute_greedy_policy(Q) # 貪欲方策を計算します  
    return policy
```

無限マルコフ決定過程での方策反復法

方策反復法は、期待ベルマン作用素やモンテカルロ法などによって Q^{π_k} を見つけ、見つけた Q^{π_k} を使って良い方策 π_{k+1} を見つけ、その繰り返しで π^* を見つける**アルゴリズム**です。

- 入力：報酬行列 r , 遷移確率行列 P , 割引率 γ , イテレーションの回数 K
 - 出力：方策 π
1. Q_0 を要素が全て0の $S \times A$ の行列で初期化する． π_0 を一様分布で初期化する．
 2. for $k = 0, 1, 2, \dots K$
 3. 方策評価： Q^{π_k} を期待ベルマン作用素やモンテカルロ法で計算する．
 4. 方策更新： Q^{π_k} の**貪欲方策**を π_{k+1} とする．
 5. π_k を出力する．

アルゴリズムが出力する方策の精度については省略します．
<https://rltheory.github.io/w2022-lecture-notes/planning-in-mdps/lec8/> などを参考にしてください．

方策反復法を実装しよう！

```
def PolicyIteration(rew, P, gamma, iteration, eval_accuracy):
    S, A = rew.shape
    Q = np.zeros((S, A)) # 適当なQの初期値を生成します
    policy = np.ones((S, A)) / A # 適当な方策の初期値を生成します
    for _ in range(iteration): # 方策更新を繰り返します

        # ベルマン期待作用素を繰り返して $Q^\pi$ を計算します
        While True:
            BQ = Bellman_expected_operator(rew, P, gamma, Q, policy)
            if np.abs(BQ - Q).max() <= eval_accuracy: # 十分な精度が期待出来たらループを抜けます
                break
            Q = BQ

        # 方策更新をします
        policy = compute_greedy_policy(Q) # 貪欲方策を計算します
    return policy
```

(補足) 強化学習と動的計画法の関係

これまでの章ではプランニング問題（ P と r が与えられた状況）について学びました。しかし、 P や r が分からない場合は、今まで学んだベルマン作用素を実装することはできません。例えばベルマン最適作用素を丁寧に書くと、

$$B(Q)(s, a) = r(s, a) + \underbrace{\sum_{s' \in S} P(s' | s, a)}_{P(\cdot | s, a) \text{ についての期待値}} \max_{a' \in A} Q(s', a')$$

であり、 $B(Q) \in \mathbb{R}^{S \times A}$ の1マスを計算するためには、 $r(s, a)$ の値と、 $P(\cdot | s, a)$ による期待値の両方を知る必要があります。しかし、特に「 $P(\cdot | s, a)$ による期待値」は現実で計算できることは稀です（ P の値は未知の場合が多いです。じゃんけんゲームを思い出しましょう。）

そこで強化学習アルゴリズムでは、厳密な期待値の計算のせずに、**サンプルを使って期待値を近似することを考えます。**

例えばQ学習であれば、サンプル $(s^{(h)}, a^{(h)}, r^{(h)}, s^{(h+1)})$ を使って

$$Q(s^{(h)}, a^{(h)}) \leftarrow (1 - \alpha)Q(s^{(h)}, a^{(h)}) + \alpha \left(r^{(h)} + \gamma \max_{a \in A} Q(s^{(h+1)}, a) \right)$$

とQ関数の行列を更新します（ $0 < \alpha < 1$ は学習率です）。

(補足) Q学習の行列表現

Q
 $S \times A$ の行列

$Q(s_0, a_0)$
$Q(s_1, a_0)$
$Q(s_2, a_0)$
$Q(s_0, a_1)$
$Q(s_1, a_1)$
$Q(s_2, a_1)$

$$\leftarrow (1 - \alpha)Q(s_1, a_0) + \alpha \left(r(s_1, a_0) + \gamma \max_{a'} Q(\underset{\substack{P(\cdot | s_1, a_0)}}{s'}, a') \right)$$

(補足) 深層強化学習と動的計画法の関係

テーブルマルコフ決定過程では行動と状態集合が有限なので、行列としてQ関数を表現できました。行動や状態集合が無限の場合、Q関数をNNなどの関数で表現することがあります。

例えば深層強化学習アルゴリズムであるDQNは、 θ でパラメータ化されたQ関数 Q_θ をデータセット \mathcal{D} を使って

$$\theta \leftarrow \operatorname{argmin}_\theta \mathbb{E}_{\mathcal{D}} \left[\left(Q_\theta(s, a) - (r + \gamma \max_{a'} Q(s', a')) \right)^2 \right]$$

と更新します。これは↓のように、無限次元のQ行列全体を、パラメータによって一気に更新しているとみなせます：

Q_θ

無限次元の
行列

...
$Q(s_1, a_0)$
$Q(s_2, a_0)$
...
...
...

$\left\{ \leftarrow \operatorname{argmin}_\theta \mathbb{E}_{\mathcal{D}} \left[\left(Q_\theta(\textcolor{red}{s}, \textcolor{red}{a}) - (\textcolor{red}{r} + \gamma \max_{a'} Q(\textcolor{red}{s}', a')) \right)^2 \right] \right.$

データセット \mathcal{D} からサンプル

3. 発展的な動的計画法

この章の目標

- この章では、価値反復法と方策反復法を**一般化した動的計画法アルゴリズム**を紹介します
これにより、価値反復法と方策反復法を別々に覚える必要がなくなります
(多くの教科書では別々に紹介していると思います)
また、オフポリシー強化学習とオンポリシー強化学習を分かりやすく理解できます
- おまけとして、動的計画法に**近似誤差が追加されたアルゴリズム**を紹介します
これは（深層）強化学習など、動的計画法を近似的に実装した形をモデル化し、アルゴリズムの設計方針を考えるときに便利です

価値反復法を書き換えよう

価値反復法を復習しましょう。次の更新を繰り返して Q^* を見つけるのが価値反復法でした。

- 価値反復法： $Q_{k+1} = r + \gamma P \left(\max_a Q_k \right)$

ここで、適当な $Q \in \mathbb{R}^{S \times A}$ について、その貪欲方策を $\text{Gr}(Q)$ と表記することにしましょう。

すると、 $\max_a Q_k = \langle \text{Gr}(Q_k), Q_k \rangle$ と同じです（貪欲方策の定義から直ちに成り立ちます。考えてみましょう）。

この $\text{Gr}: \mathbb{R}^{S \times A} \rightarrow \mathbb{R}^{S \times A}$ のことを **貪欲方策作用素** と呼ぶことにします。

よって、価値反復法は次の2つの処理の繰り返しと等価です。

- 方策更新： $\pi_{k+1} = \text{Gr}(Q_k)$
- 方策評価： $Q_{k+1} = \mathcal{B}^{\pi_{k+1}}(Q_k)$

方策反復法を書き換えよう

方策反復法は次の 2 ステップを繰り返すアルゴリズムでした。（復習です）

- 方策更新： $\pi_{k+1} = \text{Gr}(Q_k)$
- 方策評価： $Q_{k+1} = Q^{\pi_{k+1}}$

ここで、 $Q \in \mathbb{R}^{S \times A}$ に対して、 $\mathcal{B}_m^\pi: \mathbb{R}^{S \times A} \rightarrow \mathbb{R}^{S \times A}$ は、 $\mathcal{B}_m^\pi(Q) = \mathcal{B}^\pi(\mathcal{B}^\pi(\dots \mathcal{B}^\pi(Q)))$ を満たす、 Q に m 回ベルマン期待作用素を適用した出力とします。

\mathcal{B}_m^π を、**マルチステップベルマン期待作用素** と呼びます。

このとき $\lim_{m \rightarrow \infty} \mathcal{B}_m^\pi(Q) = Q^\pi$ が成り立ちます（復習です）。 よって、方策反復法は

- 方策更新： $\pi_{k+1} = \text{Gr}(Q_k)$
- 方策評価： $Q_{k+1} = \lim_{m \rightarrow \infty} \mathcal{B}_m^\pi(Q_k)$

と同じです。

マルチステップ方策評価

前ページで見たように、価値反復法と方策反復法の違いは**方策評価での B^π の適用回数**だけです。それぞれ、 B_m^π について $m = 1$ と $m = \infty$ の極端なケースを考えています。

$1 \leq m \leq \infty$ で一般化された B_m^π による方策評価を、**マルチステップ方策評価** と呼びます。

ちなみに、 B_m^π を使った反復法を**修正方策反復法 (Modified Policy Iteration)** と呼びます。

- 方策更新： $\pi_{k+1} = \text{Gr}(Q_k)$
- マルチステップ方策評価： $Q_{k+1} = B_m^\pi(Q_k)$

Modified policy iterationについての論文：<https://www.jstor.org/stable/2630487>

マルチステップ方策評価を実装しよう！

マルチステップ方策評価を実装してみましょう。ベルマン期待作用素をを繰り返すだけで実装されます。

- マルチステップ方策評価： $Q_{k+1} = \mathcal{B}_m^{\pi_{k+1}}(Q_k)$

```
def MultiStep_Bellman_expected_operator(rew, P, gamma, Q, policy, m):
    S, A = Q.shape

    # 各入力のshapeが正しいか確認します
    assert Q.shape == (S, A)
    assert policy.shape == (S, A)
    assert rew.shape == (S, A)
    assert P.shape == (S, A, S)

    for _ in range(m):
        policy_Q = (policy * Q).sum(axis=1) # <πQ>です
        P_policy_Q = P @ policy_Q # P<πQ>です
        Q = rew + gamma * P_policy_Q # ベルマン期待作用素の結果を再びQにします
    return Q
```

(補足) マルチステップ方策評価とモンテカルロ法の関係

モンテカルロ法は、マルチステップ方策評価において $m \rightarrow \infty$ の場合とみなすことができます。見てみましょう。
まずは B_1^π では

$$B_1^\pi(Q) = r + \gamma P\langle \pi, Q \rangle$$

右辺に r が 1 個。 Q が 1 個。

のように、右辺には r 一つと、 Q が 1 つ現れます。

Q学習やSARSAは、この $B_1^\pi(Q)$ をサンプルで近似します（「(補足) Q学習と動的計画法の関係」でやりましたね）。 $B_m^\pi(Q)$ では、

$$B_m^\pi(Q) = r + \gamma P\left\langle \pi, r + \gamma P\left\langle \pi, \dots \left\langle \pi, r + \gamma P Q \right\rangle \right\rangle \right\rangle$$

右辺に r が m 個。 Q が 1 個。

のように、右辺には r が m 個と、 Q が 1 つ現れます。

$m \rightarrow \infty$ の場合、

$$Q^\pi = \lim_{m \rightarrow \infty} B_m^\pi(Q) = r + \gamma P\langle \pi, r + \gamma P\langle \pi, r + \gamma P \dots \rangle \rangle$$

と r が ∞ 個現れますが、 Q が出てくることはありません。（現れても、 γ^∞ のせいで影響が 0 になります。）

モンテカルロ法は、 $\lim_{m \rightarrow \infty} B_m^\pi(Q)$ を ∞ 個の割引報酬（ $r^{(0)} + \gamma r^{(1)} + \gamma^2 r^{(2)} + \dots$ ）によって近似しています。

ここまでのまとめ

	価値反復法	方策反復法 (モンテカルロ法など)	マルチステップ 方策評価
方策更新	$\pi_{k+1} = \text{Gr}(Q_k)$	$\pi_{k+1} = \text{Gr}(Q_k)$	$\pi_{k+1} = \text{Gr}(Q_k)$
方策評価	$Q_{k+1} = \mathcal{B}_{\textcolor{red}{1}}^{\pi_{k+1}}(Q_k)$	$Q_{k+1} = \lim_{m \rightarrow \infty} \mathcal{B}_m^{\pi}(Q_k)$	$Q_{k+1} = \mathcal{B}_m^{\pi}(Q_k)$
方策評価の ステップ数	$m = 1$	$m = \infty$	m

ところで、この表では方策更新がすべて同じですね ($\pi_{k+1} = \text{Gr}(Q_k)$) .

方策更新も実はマルチステップにできます.

次ページへ

貪欲方策って良い方策なの？

これまでのアルゴリズムは、貪欲方策 $\pi_{k+1} = \text{Gr}(Q_k)$ によって方策更新をしていました。

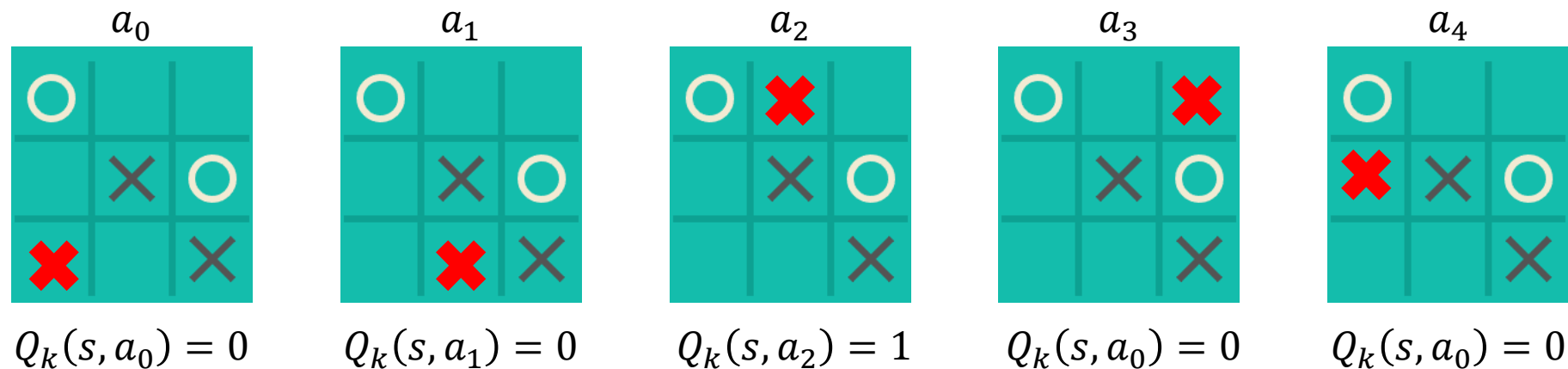
貪欲方策の利点は、その**計算効率の良さ**にあります。

(例えば1章で体験したじゃんけんゲームは $|A| = 3$ なので、 $Q(a_0, s)$, $Q(a_1, s)$, $Q(a_2, s)$ の値を比較すれば $\text{Gr}(Q)(\cdot | s)$ は求まります。)

一方で、貪欲方策 $\pi = \text{Gr}(Q)$ の期待収益 $\mathbb{E}^\pi[R]$ は、**引数の Q の精度次第で大きく変わります。**

例えば、↓の3目並べを考えましょう。 $\pi_{k+1} = \text{Gr}(Q_k)$ に従って次の **✖** を置くとします。

また、アルゴリズムは一連の処理過程で、↓のような Q_k を計算に至ったとしましょう。



このとき、 $\pi_{k+1} = \text{Gr}(Q_k)$ の貪欲方策は a_2 を決定的に選択します。

しかし、2手先まで読めば、**明らかにこの貪欲方策は勝機を逃していることがわかります** (a_0 or a_1 で勝ちが確定します)

数手先まで考えた貪欲方策：マルチステップ貪欲方策

前ページのように、貪欲方策は引数の Q の情報しか見ないため、 Q の精度が悪いと、数手先の勝利を逃すことがあります。マルチステップな貪欲方策を導入するために、次のマルチステップベルマン最適作用素を導入しましょう。

$Q \in \mathbb{R}^{S \times A}$ に対して、 $\mathcal{B}_n: \mathbb{R}^{S \times A} \rightarrow \mathbb{R}^{S \times A}$ は、 $\mathcal{B}_n(Q) = \mathcal{B}(\mathcal{B}(\dots \mathcal{B}(Q)))$ を満たす、 Q に n 回ベルマン最適作用素を適用した出力とします。 \mathcal{B}_n を、**マルチステップベルマン最適作用素**と呼びます。

\mathcal{B}_n を使うと、次の **n ステップ先まで先読みした貪欲方策**が実現できます。これを**マルチステップ貪欲方策**と呼びます。

$$\text{Gr}(\mathcal{B}_n(Q)) = \arg \max_{\pi} \left\langle \pi, r + \gamma P \left\langle \pi, r + \gamma P \left\langle \pi, \dots \left\langle r + \gamma P \langle \pi, Q \rangle \right\rangle \right\rangle \right\rangle \right\rangle$$

$0 \leq n \leq \infty$ で一般化された $\text{Gr}(\mathcal{B}_n(Q))$ による方策更新を、**マルチステップ方策更新**と呼びます。（普通の貪欲方策は $n = 0$ ですね。）

- **マルチステップ方策更新**： $\pi_{k+1} = \text{Gr}(\mathcal{B}_n(Q))$
- **マルチステップ方策評価**： $Q_{k+1} = \mathcal{B}_m(Q_k)$

（解釈）：マルチステップ貪欲方策の意義は、将棋を考えるとわかりやすいです。強い棋士は100手先までも読んだりしますが、無限手先まで読むと時間切れになってしまいます。どこかで棋士の直感に従って読みを打ち切る必要があります。マルチステップ貪欲方策における Q 値は、ここでいう棋士の直感のような役割を担っています。

マルチステップ貪欲方策を実装しよう！

マルチステップ貪欲方策を実装してみましょう。ベルマン最適作用素をを繰り返すだけで実装されます。

- マルチステップ方策更新： $\pi_{k+1} = \text{Gr}(\mathcal{B}_n(Q))$

```
def MultiStep_greedy_policy(rew, P, gamma, Q, n):  
    S, A = Q.shape  
  
    # 各入力のshapeが正しいか確認します  
    assert Q.shape == (S, A)  
    assert rew.shape == (S, A)  
    assert P.shape == (S, A, S)  
  
    for _ in range(n):  
        max_Q = Q.max(axis=1) # maxQです  
        P_max_Q = P @ max_Q # P(maxQ)です  
        Q = rew + gamma * P_max_Q #ベルマン最適作用素の結果を再びQにします  
  
    # 貪欲方策を計算します  
    policy = np.zeros((S, A))  
    policy[np.arange(S), Q.argmax(axis=1)] = 1.0  
    return policy
```

まとめ：マルチステップ動的計画法

	価値反復法	方策反復法	マルチステップ方策評価	マルチステップ 方策更新&評価 (∞)	マルチステップ 方策更新&評価
方策更新: $\pi_{k+1} =$	$\text{Gr}(Q_k)$	$\text{Gr}(Q_k)$	$\text{Gr}(Q_k)$	$\text{Gr}\left(\lim_{n \rightarrow \infty} \mathcal{B}_n(Q_k)\right)$	$\text{Gr}(\mathcal{B}_n(Q_k))$
方策評価: $Q_{k+1} =$	$\mathcal{B}_1^{\pi_{k+1}}(Q_k)$	$\lim_{m \rightarrow \infty} \mathcal{B}_m^{\pi_{k+1}}(Q_k)$	$Q_{k+1} = \mathcal{B}_m^{\pi_{k+1}}(Q_k)$	$\lim_{m \rightarrow \infty} \mathcal{B}_m^{\pi_{k+1}}(Q_k)$	$\mathcal{B}_m^{\pi_{k+1}}(Q_k)$
方策更新の ステップ数 (先読み数)	$n = 0$	$n = 0$	$n = 0$	$n = \infty$	n
方策評価の ステップ数	$m = 1$	$m = \infty$	m	$\lim_{m \rightarrow \infty} \mathcal{B}_m^{\pi}(Q_k)$	m
代表的な RLアルゴリズム	Q学習, DQN, SARSA, ...	モンテカルロ法+貪欲方策	TD3($m = 2$), Rainbow	MCTS, 方策勾配法, PPO, TRPO, 最適方策 π^* (近似なしなら)	モデル予測制御, Alpha-GO, 方策勾配法...

マルチステップ方策更新&方策評価の m, n を変えると、様々なアルゴリズムが導出されます。
また、 $m, n \geq 1$ である限り、全てのアルゴリズムが最適方策に収束します。

注：代表的なRLアルゴリズムの欄は厳密ではありません。例えばPPOの方策評価のステップ数を $m = 5$ にしても、PPO自体は動きます。
ここは「大体このあたりの枠組みに入るんだ～」くらいの感覚で確認してください。

また、TD3やPPOが一体何なのか？という話は、次の講義でやります。

補足：マルチステップとオフポリシー・オンポリシー

強化学習アルゴリズムではしばしば**オフポリシー**と**オンポリシー**という単語が出てきます。

よくある説明

- オンポリシー：方策更新（評価）の際に現在の方策のデータを使って更新する
- オフポリシー：方策更新（評価）の際にどのデータを使っても良い

例えばQ学習なら、 $Q(s^{(h)}, a^{(h)}) \leftarrow (1 - \alpha)Q(s^{(h)}, a^{(h)}) + \alpha \left(r^{(h)} + \gamma \max_{a \in A} Q(s^{(h+1)}, a) \right)$

の形で更新すると説明しましたが、これは $s^{(h)}, a^{(h)}, r^{(h)}, s^{(h+1)}$ を π で集める必要がないのでオフポリシーです。

そもそも方策更新の際に使用するデータになぜオンやオフの違いが生じるのでしょうか？

2章では「**強化学習アルゴリズムは動的計画法を近似したアルゴリズム**」と説明しました。

基本的には、強化学習アルゴリズムが近似したい動的計画法において、**方策更新と方策評価がどちらも2ステップ以上ならばオンポリシーの場合が多いです。**

次ページで詳しく解説

続き：マルチステップとオフポリシー・オンポリシー

マルチステップ方策評価を通じて、オフポリシーとオンポリシーの違いを確認しましょう。

簡単のため、 $n = 2$ の2ステップ方策評価を見てみます。

- 2ステップ方策評価： $B_n^\pi(Q) = r + \gamma P\langle \pi, r + \gamma P\langle \pi, Q \rangle \rangle$

$B_n^\pi(Q)$ の1マス(s_1, a_1)の更新を丁寧に書くと、次と同じです：

$$B_n^\pi(Q)(s_1, a_1) = r(s_1, a_1) + \gamma \sum_{s_2 \in S} P(s_2 | s_1, a_1) \sum_{a_2 \in A} \pi(a_2 | s_2) r(s_2, a_2) + \gamma^2 \sum_{s_2 \in S} P(s_2 | s_1, a_1) \sum_{a_2 \in A} \pi(a_2 | s_2) \sum_{s_3 \in S} P(s_3 | s_2, a_2) \sum_{a_3 \in A} \pi(a_3 | s_3) Q(s_3, a_3)$$

この2ステップ方策評価を強化学習で実現するには、**3項目**をサンプルで近似しなければいけません。

ここで、状態 s_3 のサンプルを得るには、「 $s_2 \sim P(\cdot | s_1, a_1)$ によって遷移した s_2 で、 $a_2 \sim \pi(\cdot | s_2)$ によって a_2 を選択し、そして、 $s_3 \sim P(\cdot | s_2, a_2)$ によってサンプルを得る」必要があります。

- つまり、 $n \geq 2$ では、 s_3 以降のサンプルを得るためには、**必ず π による意思決定と、その意思決定の結果が必要です！**
つまり、サンプルを π で集める必要があります。だからオンポリシーなのです。
- 一方で、 $n = 1$ であれば、「 $s_2 \sim P(\cdot | s_1, a_1)$ によって遷移した s_2 」までのサンプルで充分です。 a_1 は π の意思決定である必要がありません。また、 Q と π はアルゴリズムが保持している情報である場合が多く、 s_2 が分かれば、 $\sum_{a_2 \in A} \pi(a_2 | s_2) Q(s_2, a_2)$ は簡単に計算できます。だからオフポリシーなのです。

おまけ：動的計画法と近似誤差

第二章で、「（深層）強化学習アルゴリズムは動的計画法の近似とみなせる」，という説明をしました。すなわち次のように，動的計画法の更新で近似誤差 $\epsilon_k \in \mathbb{R}^{S \times A}$ が生じるとみなせます。

- 方策更新： $\pi_{k+1} = \text{Gr}(Q_k)$
- 方策評価： $Q_{k+1} = \mathcal{B}^{\pi_{k+1}}(Q_k) + \epsilon_k$

このように近似誤差が追加された動的計画法を，**近似動的計画法**と呼びます。特に上の更新は**近似価値反復法（Approximate Value Iteration）**と呼ばれます。

近似価値反復法の分かりやすい論文：<https://arxiv.org/abs/2003.14089>

（簡単のために方策評価にだけ近似誤差を追加しました。DQNなどは方策更新には誤差がでません。しかし例えば連続値制御の場合，方策更新にも誤差が生じます。また，簡単のために1ステップアルゴリズムを考えています。マルチステップの結果は調べてみてください。）

おまけ：価値反復法と誤差頑健性

価値反復法の目標は「 Q^{π^*} を見つけること」であり、ベルマン最適作用素の収束性から、価値反復法では

$\lim_{k \rightarrow \infty} Q_k = Q^{\pi^*}$ が成立します。では、次のように更新時に誤差が生じた場合、価値反復法は収束するのでしょうか？

- 方策更新： $\pi_{k+1} = \text{Gr}(Q_k)$
- 方策評価： $Q_{k+1} = \mathcal{B}^{\pi_{k+1}}(Q_k) + \epsilon_k$

上の近似価値反復法において、次の誤差バウンドが成立します。

$$\|Q^{\pi^*} - Q^{\pi_k}\|_{\infty} \leq \frac{2}{(1-\gamma)^2} \max_{0 \leq j \leq k} \|\epsilon_j\| + \frac{\gamma^k}{1-\gamma}$$

- 誤差が全くない場合、**1項目**は0です。つまり、 $\|Q^{\pi^*} - Q^{\pi_k}\|_{\infty} \leq \frac{\gamma^k}{1-\gamma}$ であり、 π_k は指数オーダで π^* に近づきます。
- 誤差がある場合、 $\frac{2}{(1-\gamma)^2} \max_{0 \leq j \leq k} \|\epsilon_j\|$ は、「過去に生じた最大の誤差が $\frac{2}{(1-\gamma)^2}$ 倍されて性能に影響する」ことを表しています。
例えば $\gamma = 0.99$ の場合などは誤差の影響が1万倍されることになります。
これは価値反復法が誤差に弱いことを示しています。

おまけ：誤差頑健性と正則化

強化学習ではしばしばKL正則化を利用したアルゴリズムを考えます。（TRPOやPPO）

実はKL正則化を入れると、誤差に対して頑健になることが知られています。

- 方策更新： $\pi_{k+1} = \operatorname{argmax}_{\pi} \langle \pi, Q_k \rangle - \lambda \cdot \text{KL}(\pi \parallel \pi_k)$

π_{k+1} と π_k が離れすぎないように方策更新します。

- 方策評価： $Q_{k+1} = r + \gamma P(\langle \pi_{k+1}, q \rangle - \lambda \cdot \text{KL}(\pi_{k+1} \parallel \pi_k)) + \epsilon_k$

π_{k+1} と π_k が離れすぎると方策の価値が低くなるようにします。

このようにKL正則化を導入すると、次の誤差バウンドが成立します。

$$\|Q^{\pi^*} - Q^{\pi_k}\|_{\infty} \leq \frac{2}{(1-\gamma)} \left\| \frac{1}{k} \sum_{j=0}^k \epsilon_j \right\| + \frac{4}{(1-\gamma)^2 \cdot k}$$

- 誤差が全くない場合、1項目は0です。つまり、 $\|Q^{\pi^*} - Q^{\pi_k}\|_{\infty} \leq \frac{4}{(1-\gamma)^2 \cdot k}$ であり、 π_k は線形オードで π^* に近づきます。
- 誤差がある場合、 $\frac{2}{(1-\gamma)} \left\| \frac{1}{k} \sum_{j=0}^k \epsilon_j \right\|$ は、「過去に生じた誤差の平均値が $\frac{2}{(1-\gamma)}$ 倍されて性能に影響する」ことを表しています。
例えば期待値が0の誤差が追加される場合は1項目も0であり、誤差の存在下でも収束が保証されます。
これは、**KL正則化を入れると価値反復法が誤差に強くなることを示しています。**

まとめ

- 第一章：問題設定の定義
 - 逐次意思決定問題
 - テーブルマルコフ決定過程
 - 有限ホライゾン，無限ホライゾン
- 第二章：最適方策の求め方
 - 動的計画法
 - 価値反復法・方策反復法
- 第三章：発展的な動的計画法
 - 動的計画法のマルチステップ化
 - おまけ：正則化と誤差頑健性