Value Iteration

Joshua Tsang

September 14, 2024

Contents

1 Foundational Concepts

It is instructive intially discuss the foundational concepts in Markov Decision Problems (MDP) and Reinforcement Learning (RL). Consider the problem shown in Figure 1 where an agent is located in one of the grid positions.

- State Space and State: The finite State Space of a system is denoted S where a state s denotes a certain configuration of the system i.e. $s \in S$. For example, in Figure 1 the agent is currently in state s_4, with the neighbouring states, $\{s'\}$, being s_3 and s_5.

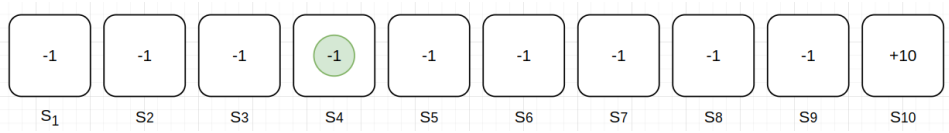

Figure 1: 1D Grid World MDP with an assigned reward of +10 in state s_{10}, termed the terminal state. The agent currently sits in state s_4 and the goal of the problem is for the agent to take actions to maximise its reward. At each state other than the terminal state, there is a negative reward of -1. Drawn with draw.io and saved in Google Drive as 1D_grid_world.drawio.

- Actions: For each state, s_i, that the system resides in a set of available actions, $\{a_k\}$, can be taken that transition the system to another state, s_j. For Figure 1 the available actions at state s_4 are `(move left)` and `(move right)`. One could think of a graph where nodes are states and the edges are actions.

- Policy: A policy, denoted π, is a distribution over actions given states. It is often written as a function $\pi(a|s)$ or $\pi(s, a)$ where it is formally defined as:

$$\pi(a|s) = \pi(s, a) = P(A_t = a|S_t = s) \tag{1}$$

  which is the probability of taking action a given the system is in state s at time step t. As such, $\pi(s, a)$ returns a probability value in the range $[0, 1]$. The goal of RL is to learn the optimal policy $\pi(s, a)$ to maximise the discounted sum of future rewards/returns (explained next). As a foreshadow, *optimal* policy functions are often expressed as an argmax_a as follows:

$$\begin{aligned} \pi_*(s, a) &= \underset{a}{\text{argmax}} \sum_{s'} P(s', r|s, a)[r + \gamma v_\pi(s')] \\ &= \underset{a}{\text{argmax}}\, q_*(s, a) \end{aligned} \tag{2}$$

  i.e. the optimal policy function returns the action a that transitions the agent to the neighbouring state s' that yields the maximum transition-probability-weighted sum of of future rewards, or in other words, the maximum *state-action value* as discussed around equation (14).

- Reward: Numerical reward can be associated with certain states and actions, represented as R_t for the current time step. In the case of Figure 1, there is a fixed reward of +10 at state s_{10} and the other states have a reward of -1. The goal of RL can be considered to be finding the sequence of actions that maximise the sum of rewards to achieve a sucessful episode i.e. the agent reaches the terminal state s_{10}.

- Episodes: MDPs have the characteristic of being finite i.e. they end within a finite number of time steps, denoted T. More specifically, an episode is a sequence of states, actions and rewards that start at some state and end in the terminal state.

- Return or "Future Gains": Usually denoted G_t it is the sum of rewards gained by the agent *after* time step t, define as:

$$G_t = R_{t+1} + R_{t+2} + ... + R_T \tag{3}$$

Before going further, it's worth appreciating how important this concept is to the definition of state value functions, $v(s)$, which try to capture the expected rewards to be gained by being in that state, s.

It is possible to introduce a discount rate $\gamma \in [0, 1]$ that diminishes the rewards gained further in the future:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-1-t} R_T \\ &= \sum_{k=t+1}^{T} \gamma^{k-1-t} R_k \end{aligned} \tag{4}$$

note how the power of γ in the final term is $T-1-t$ and *not* T, a little thought makes this clear (for example, try write down G_4 where $T = 8$). Finally, we note that the discounted expression can be rewritten into a recursive expression somewhat similar to Bellman's equation:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-1-t} R_T \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ... + \gamma^{T-t} R_T) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{5}$$

Expression (5) links the current return, G_t, to the return at the next time step, G_{t+1}.

1.1 Markov Reward Process vs Markov Decision Process

It is instructive to discuss Markov Reward Process (MRP) which are simpler than a Markov Decision Process (MDP). This discussion helps consolidate concepts surrounding transition probabilites, policies and action values. In short, an MDP is essentially an MRP *with actions* and thus an MRP will still have states, rewards and transition probabilties. The transition probabilities encapsulate the dynamics of the process or the effects of the environment.

Consider the MRP shown in Figure 2 which describes a 'slippery world' MRP where the transition probabilities of moving to adjacent states is only succesful with probability 0.8, with a 0.2 chance of remaining in the same state.

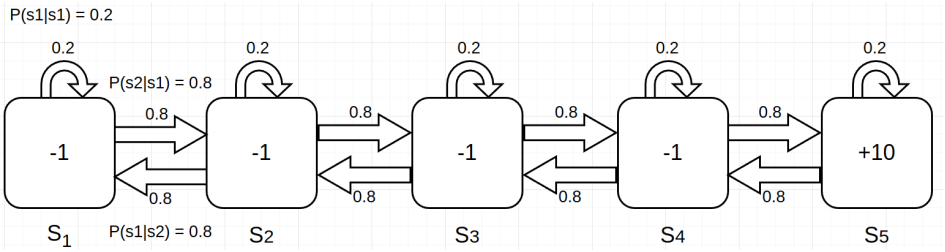

Figure 2: The Slippery World MRP where there's a 0.2 chance of remaining in the present state as the agent can slip and fall, making no movement progress. There are still rewards associated with the states, hence the name Markov *Reward* Process, but no actions so it's not a Markov *Decision* Process. Drawn with draw.io and saved in Google Drive.

The state transition matrix, $\mathbf{P}$, for the MRP can be written:

$$\begin{aligned}\mathbf{P} &= \begin{pmatrix} P(s_1|s_1) & P(s_2|s_1) & P(s_3|s_1) & \dots & P(s_N|s_1) \\ P(s_1|s_2) & P(s_2|s_2) & P(s_3|s_2) & \dots & P(s_N|s_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ P(s_1|s_N) & P(s_2|s_N) & P(s_3|s_N) & \dots & P(s_N|s_N) \end{pmatrix} \\ &= \begin{pmatrix} 0.2 & 0.8 & 0.0 & 0.0 & 0.0 \\ 0.8 & 0.2 & 0.8 & 0.0 & 0.0 \\ 0.0 & 0.8 & 0.2 & 0.8 & 0.0 \\ 0.0 & 0.0 & 0.8 & 0.2 & 0.8 \\ 0.0 & 0.0 & 0.0 & 0.8 & 0.2 \end{pmatrix}\end{aligned} \tag{6}$$

Knowing these state transition probabilties allows trajectories to be sampled and expect returns to be calculated. Like an MDP, state values for each state, $v(s)$, can be computed. In fact, the state values can be computed analytically using the linear equation:

$$\begin{aligned}\begin{pmatrix} v(s_1) \\ v(s_2) \\ \vdots \\ v(s_5) \end{pmatrix} &= \begin{pmatrix} r(s_1) \\ r(s_2) \\ \vdots \\ r(s_5) \end{pmatrix} + \gamma \begin{pmatrix} P(s_1|s_1) & P(s_2|s_1) & P(s_3|s_1) & \dots & P(s_5|s_1) \\ P(s_1|s_2) & P(s_2|s_2) & P(s_3|s_2) & \dots & P(s_5|s_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ P(s_1|s_5) & P(s_2|s_5) & P(s_3|s_5) & \dots & P(s_5|s_5) \end{pmatrix} \begin{pmatrix} v(s_1) \\ v(s_2) \\ \vdots \\ v(s_5) \end{pmatrix} \\ \mathbf{v} &= \mathbf{r} + \gamma \mathbf{P} \mathbf{v}\end{aligned} \tag{7}$$

which allows the state values to be solved using some linear algebra as:

$$\mathbf{v} = (\mathbf{I} - \gamma \mathbf{P})^{-1} \mathbf{r} \tag{8}$$

Solving the matrix is roughly $O(N^3)$ where N is the number of states.

2 Playing with Episodes of an MDP

Recall that epsiodes are a sequence of states, actions and rewards that end in the terminal state. To clarify concepts and develop some intuitive, consider a few episodes for the Grid World system in Figure 1. The actions are $a \in [\text{LEFT}, \text{RIGHT}]$

- Episode Example 1: A simple episode where the agent walks directly to the terminal state.

  $[t = 1, S_1 = s_4, A_1 = \text{RIGHT}, R_1 = -1, G_1 = R_2 + \ldots + R_7 = +5]$
  $[t = 2, S_2 = s_5, A_2 = \text{RIGHT}, R_2 = -1, G_2 = R_3 + \ldots + R_7 = +6]$
  $[t = 3, S_3 = s_6, A_3 = \text{RIGHT}, R_3 = -1, G_3 = R_4 + \ldots + R_7 = +7]$
  $[t = 4, S_4 = s_7, A_4 = \text{RIGHT}, R_4 = -1, G_4 = R_5 + \ldots + R_7 = +8]$
  $[t = 5, S_5 = s_8, A_5 = \text{RIGHT}, R_5 = -1, G_5 = R_6 + R_7 = +9]$
  $[t = 6, S_6 = s_9, A_6 = \text{RIGHT}, R_6 = -1, G_6 = R_7 = +10]$
  $[t = 7, S_7 = s_{10}, A_7 = \text{RIGHT}, R_7 = +10, G_7 = NA]$

  Note how the future returns, G_t, for each state can be calculated using Equation 3.

- Episode Example 2:

  $[t = 1, S_1 = s_4, A_1 = \text{RIGHT}, R_1 = -1, G_1 = R_2 + \ldots + R_9 = +3]$
  $[t = 2, S_2 = s_5, A_2 = \text{RIGHT}, R_2 = -1, G_2 = R_3 + \ldots + R_9 = +4]$
  $[t = 3, S_3 = s_6, A_3 = \text{RIGHT}, R_3 = -1, G_3 = R_4 + \ldots + R_9 = +5]$
  $[t = 4, S_4 = s_5, A_4 = \text{LEFT}, R_4 = -1, G_4 = R_5 + \ldots + R_9 = +6]$
  $[t = 5, S_5 = s_6, A_5 = \text{RIGHT}, R_5 = -1, G_5 = R_6 + \ldots + R_9 = +7]$
  $[t = 6, S_6 = s_7, A_6 = \text{RIGHT}, R_6 = -1, G_6 = R_7 + \ldots + R_9 = +8]$
  $[t = 7, S_7 = s_8, A_7 = \text{RIGHT}, R_7 = -1, G_7 = R_8 + R_9 = +9]$
  $[t = 8, S_8 = s_9, A_8 = \text{RIGHT}, R_8 = -1, G_8 = R_9 = +10]$
  $[t = 9, S_9 = s_{10}, A_9 = \text{RIGHT}, R_9 = +10, G_9 = NA]$

  Note that this episode backtracks by going LEFT at $t = 4$ which caused it to require a longer overall episode.

3 State Value Function and Bellman's Equation

An important concept in RL is the state value function, $v_\pi(s)$, which captures the expected return from being in state s. It is critical to realise that the subscript π in $v_\pi(s)$ means it must be evaluated for a given policy $\pi(a|s)$ since $v_\pi(s)$ is an expectation value (read: average) of potential returns by being in state s, which in turn depends on the expected return from the neighbouring states, $v_\pi(s')$. In fact, this is why the Bellman equation in equation (11) has $v_\pi(s)$ on the LHS and $v_\pi(s')$'s on the RHS.

Since the policy $\pi(a|s)$ encapsulates the current policy's probability of taking action a given the current state is s (recall equation 1), it thus influences the expected returns of being in the current state s. In fact, it's prudent to think of the state value function as a function that ultimately links the state values of all states together.

The state value function, $v_\pi(s)$ of an MDP is the expected return starting from state s and following the policy π:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[G_t|s] \\ &= \sum_a \pi(a|s)\mathbb{E}[G_t|s,a] \end{aligned} \tag{9}$$

This is fairly intuitive, the state value is equal to the sum of the expected returns from all available actions a at state s weighted by the present policy probability distribution, π.

Using Equation (5) we can rewrite $\mathbb{E}[G_t|s,a]$ as:

$$\begin{aligned} \mathbb{E}[G_t|s,a] &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|s,a] \\ &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|s,a] \\ &= \sum_{s'} P(s',r|s,a)[r + \gamma v_\pi(s')] \end{aligned} \tag{10}$$

This is in fact the action value function as discussed in the next section. The final expression after substitution back into equation (9) is:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s',r|s,a)[r + \gamma v_\pi(s')] \tag{11}$$

Equation (11) allows us to calculate the expected return value for each state in the MDP given the current policy, π, which may not yet be optimal. We haven't yet discussed optimal policies at all and equation (11) simply calculates state values for a given policy $\pi(a|s)$. It is the goal of RL to find the optimal policy, π_*, via methods like General Policy Iteration (GPI), Policy Iteration or Value Iteration to discover optimal policies.

Note that the probabilities $P(s',r|s,a)$ are not to be confused with the policy $\pi(s,a)$ but are transition probabilities intrinsic to the MDP that capture the dynamics of the environment. Note that for an MRP the state transition probabilties are merely $P(s'|s)$ while for an MDP (which includes actions) it is $P(s',r|s,a)$ which encapsulates probabilties to transition into s' for a given action a. For instance, we could introduce a strong fan effect that blows left in state s_4 of the MDP in Figure 1 where there's now a 0.2 chance of the agent losing its footing/grip and will be blown back to state s_3. These envionmental dynamics/effects are captured in the transition probabilities of the MDP. So in this case, if we're in state s_4 then $P(s_5,r|s_4,a=\text{RIGHT}) = 0.8$ and $P(s_3,r|s_4,a=\text{RIGHT}) = 0.2$ i.e. there's a 0.2 chance of transitioning into the state in the opposite direction. The

goal of RL is to deduce the optimal policy taking into account these environmental/dynamic effects.

Now is a good time to mention model-based and model-free processes. A model-based process has full knowledge of the transition probabilities $P(s', r|s, a)$, which although introduce a probabalistic element to the process, these transitions are known and the MDP is well-defined.

(Talk about Gambler's problem and the transition probabilities in that.)

3.1 Evaluating the state value function for an example MDP

Consider the 'Slippery World' MRP shown in Figure 2 and consider that we promote it to an MDP complete with actions. At each state, there are 2 actions $a \in \{a_1, a_2\}$ where a_1 and a_2 represent the actions move LEFT and move RIGHT respectively. Since the MDP has $S = 5$ states and each state has $A = 2$ actions, the total number of possible policies is $A^S = 2^5 = 32$. Each state only has one 'good' action so one can imagine each policy being like a binary sequence: 01110 for example means $[a_1, a_2, a_2, a_2, a_1]$ and it's easy to imagine there are 32 possible combinations of these actions.

Let's try and evaluate the state value function equation (11) for state s_4 given the policy defined as follows:

$$\begin{bmatrix} (\pi(a_1|s_1), \pi(a_2|s_1)) \\ (\pi(a_1|s_2), \pi(a_2|s_2)) \\ (\pi(a_1|s_3), \pi(a_2|s_3)) \\ (\pi(a_1|s_4), \pi(a_2|s_4)) \\ (\pi(a_1|s_5), \pi(a_2|s_5)) \end{bmatrix} = \begin{bmatrix} (0.4, 0.6) \\ (0.4, 0.6) \\ (0.4, 0.6) \\ (0.4, 0.6) \\ (0.4, 0.6) \end{bmatrix} \tag{12}$$

So the policy here is a 0.4 and 0.6 chance of taking action to move LEFT and RIGHT respectively, it effectively tries to do a 40/60 random walk in this slippery world but note it might not be able to actually do a perfect random walk due to the slippery dynamics encoded into the transition probabilities $P(s', r|s, a)$. Computing $v_\pi(s_4)$ with a discount factor of $\gamma = 0.5$:

$$\begin{aligned} v_\pi(s_4) &= \pi(a_1|s_4)[P(s_4, r|s_4, a_1)(r + \gamma v_\pi(s_4)) + P(s_3, r|s_3, a_1)(r + \gamma v_\pi(s_3))] \\ &\quad + \pi(a_2|s_4)[P(s_4, r|s_4, a_2)(r + \gamma v_\pi(s_4)) + P(s_5, r|s_4, a_2)(r + \gamma v_\pi(s_5))] \\ &= 0.4[0.2(-1 + 0.5 \times -1) + 0.8(-1 + 0.5 \times -1)] \\ &\quad + 0.6[0.2(-1 + 0.5 \times -1) + 0.8(+10 + 0.5 \times +10)] \\ &= -0.36 + 7.38 \\ &= 7.02 \end{aligned} \tag{13}$$

A similar computation can be performed for all states in the MDP. Note that a different policy π would give a different value for $v_\pi(s_4)$. For example, if $\pi(a_1|s_4) = 1.0$ and $\pi(a_2|s_4) = 0.0$ then none of s_5's significant +10 reward

will be included in the value of $v_\pi(s_4)$ yielding a much lower state value. It is clear that for each state, there exists a policy that would maximise their state values. For s_4 it is clear to let $\pi(a_1|s_4) = 0.0$ and $\pi(a_2|s_4) = 1.0$ so the agent always tries to move RIGHT to claim that significant +10 reward.

Note what might happen through repeated applications/sweeps of equation (11) to all the states of the MDP. It intuitive to see that the fixed +10 reward in state s_5 will gradually 'trickle' down into the state values of the states on the left.

4 State-Action Value Function, Q-Value Tables and Optimal Policies

The action value function, denoted q_π, is defined as the inner sums in Equation (11):

$$q_\pi(s, a) = \sum_{s'} P(s', r|s, a)[r + \gamma v_\pi(s')] \tag{14}$$

where s' are the neighbouring states to the current state, s. Do not confuse state values functions and state-action value functions. State-action value functions include the action in the argument, naturally! Equation (14) allows the optimal policy, π_*, to be written succinctly as:

$$\pi_*(s) = \underset{a}{\text{argmax}}\, q_\pi(s, a) \tag{15}$$

You will notice that 'optimal' expressions usually have some form of $\max_a$ or argmax_a in it.

It is instructive to represent Equation (14) computed as a matrix for a particular MDP in the following manner:

$$\begin{bmatrix} q(s_1, a_1) & q(s_1, a_2) & \dots & q(s_1, a_K) \\ q(s_2, a_1) & q(s_2, a_2) & \dots & q(s_2, a_K) \\ \vdots & \vdots & \ddots & \vdots \\ q(s_N, a_1) & q(s_N, a_2) & \dots & q(s_N, a_K) \end{bmatrix} \tag{16}$$

where N is the number of states and K is the number of actions per state. Matrix (16) is often referred to as a 'Q-value table'.

As a foreshadow, Q-Learning attempts to construct this Q-value table which is used by the agent to determine the optimal action for a given state by selecting the action that yields the maximum q value. So say for s_4 in the Slippery World MDP, the agent will look at the row for s_4 and pick the action a (column) that yields the greatest $q(s_4, a)$ value. You may wonder how to compute the Q-value table if we don't know the optimal policy. The exact details of how the Q-value table is constructed will not be explicitly stated here, but note that Q-Learning is an *off-policy* learning scheme and thus exploration of the state space is done by a behaviour policy, $b(a|s)$, instead of $\pi(a|s)$ in the case of on-policy schemes.

5 Policy Evaluation

Policy evaluation, policy iteration and value iteration are all interconnected ideas. Policy evaluation is perhaps the most "atomic" of them all and it's the process of computing the value function, $v_\pi(s)$, for all states in an MDP for a given policy, π. It is essentially the iteration application of the Bellman expectation equation (11), which is rewritten here for convenience:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s', r|s, a)[r + \gamma v_\pi(s')] \tag{17}$$

The application of equation (17) is executed in the following manner:

- Define a policy $\pi(a|s)$. For instance, in the 1D Grid World example, we can pick a 'uniform random' policy where:

$$\begin{aligned} \pi(a = a_1|s = *) &= 0.5 \\ \pi(a = a_2|s = *) &= 0.5 \end{aligned} \tag{18}$$

  The policy is just a singular simple example. In general, any policy can be defined such as:

$$\begin{aligned} \pi(a = a_1|s = s_1) &= 0.0 \\ \pi(a = a_2|s = s_1) &= 1.0 \\ \pi(a = a_1|s = s_2) &= 0.7 \\ \pi(a = a_2|s = s_2) &= 0.3 \\ &\vdots \\ \pi(a = a_1|s = s_9) &= 0.4 \\ \pi(a = a_2|s = s_9) &= 0.6 \end{aligned} \tag{19}$$

  and equation (17) can be used to compute the state values, $v_\pi(s)$ for each state.

- At each iteration $k + 1$
- For all states $s \in S$
- Update $v_{k+1}(s)$ from $v_k(s')$ using Bellman expectation equation: This means we compute an improved estimate of the state value $v_{k+1}(s)$ based on the state values from the previous iteration k, $v_k(s')$. Explicitly, it is the Bellman expectation equation (17):

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s'} P(s', r|s, a)[r + \gamma v_k(s')] \tag{20}$$

- Repeat until convergence is reached, in which case $v_{k+1}(s) = v_\pi(s)$.

Note that this algorithm only converges to the true state values for our given policy π, it isn't computing anything relating to the optimal policy $v_*(s)$ (shorthand of $v_{\pi_*}(s)$) yet. Optimal policies are found by policy iteration or value iteration, which both use policy evaluation but then modify the policy in some way after successive policy evaluations phases. Somewhat confusingly, in David Silver's lectures, policy evaluation of the uniform random policy in 2D Grid World yields conveged state values from which a sensible policy can be extracted by $\text{argmax}_a\, v_\pi(s)$, but I suspect this only happened because of the uniform random policy and the somewhat homogenous reward environment.

6 Policy Iteration

Policy evaluation is strictly an *evaluation* process for a given policy and does not suggest how to improve that policy per se. Policy iteration is a strategy to find the optimal policy for a given MDP. It involes an alternating process of policy evaluation (as described above) and greedy policy improvement.

- Evaluate a given policy, π, as per Section 5 i.e. compute the state values $v_\pi(s)$ for all states for a given policy π.
- Modify the policy by making it act greedily with respect to v_π. The new policy is $\pi'(s)$:

$$\pi'(s) = \underset{a}{\text{argmax}}\, q_\pi(s, a) \tag{21}$$

  Note how similar the above definition is to the expression for picking the optimal action from the Q-table in equation (15).

- Repeat policy evaluation and policy improvement until convergence to an optimal policy.

One can start with any 'random' policy and for an MDP this process is guaranteed to converge towards the optimal policy v_*. The starting policy could be a *stochastic* policy akin to equations (19), or a *deterministic* policy where $\pi(a|s) \in \{0, 1\}$ i.e. $\pi(a|s)$ maps each state $s \in S$ to a single action $a \in A$. It is clear that Policy Iteration will yield a deterministic optimal policy, and for MDPs it is known that the optimal policy will be deterministic. However, there are cases where a stochastic policy is needed, such as in a game of rock, paper, scissors. I believe stochastic policy can be derived from policy gradient methods, but I'm not sure at the moment.

```
Iteration =  0, V_new = {1: -1.0, 2: -1.0, 3: -1.0, 4: -1.0, 5: -1.0, 6: -1.0, 7: -1.0, 8: -1.0, 9: -1.0, 10: 1.0}
Iteration =  1, V_new = {1: -2.0, 2: -2.0, 3: -2.0, 4: -2.0, 5: -2.0, 6: -2.0, 7: -2.0, 8: -2.0, 9: 0.0, 10: 1.0}
Iteration =  2, V_new = {1: -3.0, 2: -3.0, 3: -3.0, 4: -3.0, 5: -3.0, 6: -3.0, 7: -3.0, 8: -1.0, 9: 0.0, 10: 1.0}
Iteration =  3, V_new = {1: -4.0, 2: -4.0, 3: -4.0, 4: -4.0, 5: -4.0, 6: -4.0, 7: -2.0, 8: -1.0, 9: 0.0, 10: 1.0}
Iteration =  4, V_new = {1: -5.0, 2: -5.0, 3: -5.0, 4: -5.0, 5: -5.0, 6: -3.0, 7: -2.0, 8: -1.0, 9: 0.0, 10: 1.0}
Iteration =  5, V_new = {1: -6.0, 2: -6.0, 3: -6.0, 4: -6.0, 5: -4.0, 6: -3.0, 7: -2.0, 8: -1.0, 9: 0.0, 10: 1.0}
Iteration =  6, V_new = {1: -7.0, 2: -7.0, 3: -7.0, 4: -5.0, 5: -4.0, 6: -3.0, 7: -2.0, 8: -1.0, 9: 0.0, 10: 1.0}
Iteration =  7, V_new = {1: -8.0, 2: -8.0, 3: -6.0, 4: -5.0, 5: -4.0, 6: -3.0, 7: -2.0, 8: -1.0, 9: 0.0, 10: 1.0}
Iteration =  8, V_new = {1: -9.0, 2: -7.0, 3: -6.0, 4: -5.0, 5: -4.0, 6: -3.0, 7: -2.0, 8: -1.0, 9: 0.0, 10: 1.0}
Iteration =  9, V_new = {1: -8.0, 2: -7.0, 3: -6.0, 4: -5.0, 5: -4.0, 6: -3.0, 7: -2.0, 8: -1.0, 9: 0.0, 10: 1.0}
Iteration = 10, V_new = {1: -8.0, 2: -7.0, 3: -6.0, 4: -5.0, 5: -4.0, 6: -3.0, 7: -2.0, 8: -1.0, 9: 0.0, 10: 1.0}
Iteration = 11, V_new = {1: -8.0, 2: -7.0, 3: -6.0, 4: -5.0, 5: -4.0, 6: -3.0, 7: -2.0, 8: -1.0, 9: 0.0, 10: 1.0}
```

Figure 3: Iterations of the value iteration algorithm.

7 Value Iteration

Value iteration is another mechanism for finding the optimal policy, π_*. Recall that for policy evaluation, the Bellman *expectation* equation wwas used to find the value function for a given policy, v_π. For value *iteration*, one uses an iterative application of the Bellman *optimality* equation to find optimal value function. Note that value iteration does not explicitly build an improved policy at each step like policy evaluation.

- At each iteration $k+1$
- For all states $s \in S$
- Update $v_{k+1}(s)$ from $v_k(s')$ using the Bellman optimality equation:

$$v_{k+1}(s) = \max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} P(s', r|s, a) v_k(s') \right] \tag{22}$$

  Note how the state transition probability function/matrix $P(s', r|s, a)$ no longer weighs the reward r. I'm not complete sure why this is, but this is the expression in David Silver's lectures. In the script *line_value_iteration_1.py*, $P(s', r|s, a)$ is used to weigh the reward also.

Figure 3 displays the output of running the value iteration script *line_value_iteration_1.py*.

8 Model-Free Learning Schemes

Brilliant explanation in video 3 Mutual Information youtube video. Key point is that $P(s', r|s, a)$ in Equation (14) is often unavailable in real-life practical problems and so we directly estimate the $q_*(s, a)$ by associating the final returns of episodic trajectories to state values (estimate $v(s)$) or state-action pairs (estimate $q(s, a)$).

9 On-Policy and Off-Policy Learning

For on-policy schemes the policy $\pi(s|a)$ is used to explore the state space:

$$\pi(a|s) = b(a|s) \tag{23}$$

while for off-policy schemes e.g. Q-Learning, Off-Policy Monte Carlo etc. the roles are separated with exploration done by a separate behaviour policy, $b(a|s)$,:

$$\pi(a|s) \neq b(a|s) \tag{24}$$

Since the sampling of trajectories is done using $b(a|s)$ instead of $\pi(a|s)$, importance sampling is used to rescale the sampled distributions to recover return contributions as if the sampled was done by $\pi(a|s)$. This is similar to Umbrella Sampling in computational materials science. One can initialise trajectories from any state and explore using the behaviour policy $b(a|s)$ and generate numerous trajectories and their associated returns. Then use a credit assignment scheme like Monte Carlo Learning to assign values to either the state value function $v(s)$ of each state, or state-action values $q(s, a)$ to each state-action pair.