Theory 1 Assuming that the value functions are monotonically increasing by applying the Bellman operator, it holds that

$$Tv^{\pi} \leq v^{\pi}$$

and so, starting from zero and after k times

$$T^k v^{\pi_0} \leq v^{\pi k}$$

and since the value function is monotonic and we admit the existance of an optimal value function it holds that

$$v^{\pi k} \leq v^*$$

But that means also that

$$v^* - v^{\pi k} \leq v^* - T^k v^{\pi_0}$$

Taking infinity norms left and right

$$||v^* - v^{\pi k}||_\infty \leq ||v^* - T^k v^{\pi_0}||_\infty$$

We can apply the concept of fixed point, which means that applying the Bellman operator to an already optimal value function doesn't change it, so it holds that

$$||v^* - T^k v^{\pi_0}||_\infty = ||Tv^* - T^k v^{\pi_0}||_\infty$$

And since, the Bellman operator is a contraction, it finally holds that

$$||Tv^* - T^k v^{\pi_0}||_\infty \leq \gamma^k ||v^* - v^{\pi_0}||_\infty$$

Which means we have demonstrated that we converge to the optimal value function because, very intuitively, the distance between current policy and optimal policy becomes smaller after k iterations, so

$$||v^* - v^{\pi k}||_\infty \leq \gamma^k ||v^* - v^{\pi_0}||_\infty$$

Theory 2 This problem requires to evaluate the value function using the update rule applied to the Bellman equation. Intuitively, the value at iteration $k+1$ for every state s is obtained from the values at iteration k of every successor state s' and the immediate reward, everything weighted by the probabilities defined by the transition function.
Since the policy π yields action a_1 for every state, the formula will be

$$v^{\pi}_{k+1}(s) = \sum_{s',r} p(s', r|s, a)[r + \gamma v^{\pi}_k(s')]$$

In the case $s = s_6$, it holds that (omitting π in the formula)

$$v_{k+1}(s_6) = p(s_6|s_6, a_1)[r + \gamma v_k(s_6)] + p(s_7|s_6, a_1)[r + \gamma v_k(s_7)]$$

Now, the first reward is certainly zero, since it is a transition from s_6 to s_6. Concerning the second reward though, there was some discussion with my colleages: if the reward function $r(s, a) = 10 if s = s_7$ would be considered as s_7 the current state and a any action from that state, than obviously the second reward is zero, since the reward considered in the second member of the sum is referred to the transition from s_6 to s_7 and it would hold that

$$0.5[0 + 0.5 \cdot 0] + 0.5[0 + 0.5 \cdot 10] = 2.5$$

If, instead, the reward is considered to be obtained if we land in s_7, than it would hold that

$$0.5[0 + 0.5 \cdot 0] + 0.5[10 + 0.5 \cdot 10] = 7.5$$

Practice 1 The goal of this practical exercise is to implement the policy iteration algorithm on the grid world environment. The algorithm can be subdivided in two phases. The first one is policy evaluation, which is done with a while loop, with guard condition a delta that checks if a value function is changing "less enough" between an iteration and the other. In every iteration of the while, I cycle over all the states, calculating the value function according to the current policy. As soon as the delta is less than an epsilon, the policy has been succesfully evaluated and we can proceed with the second phase, which consists in improving the policy. In this phase the policy tries to be greedy wrt the value function, selecting the action that maximises it for every state. So basically these two phases are alternated back and forth and everything is wrapped into an external for cycle with a maximum of iterations. In the end this for cycle will end as soon as we have reached convergence for the policy, which means we have reached the optimal policy. his last condition is checked with the boolean `policy_stable`, which is set true at the beginning every external for iteration: if at the end of policy evaluation and policy improvement, nothing has changed, we reached convergence. It is important to note that policy iteration manages to converge with very few iterations (5 in one episode).

Practice 2 The goal of this exercise is to implement le iLQR algorithm applied to the cartpole environment. Since it is a dynamical system, first thing is to obtain the the matrices A and B as from $x_{t+1} = Ax_t + Bu_t$. I choose Sympy to calculate the matrices with the function `f.diff(x)`, where f is the function describing the dynamical system and x is the vector representing the state. Once I obtained the matrices, I simply copy them into the two functions

`getA` and `getB`. Next thing is to define dimensions of the two matrices Q and R that contribute to the error function (and are positive definite) and of the matrix P which we assume contributes to the optimal value function. Naturally, the matrices' dimensions will be defined according to $d = 4$, dimension of the state, ad $k = 1$, dimension of the input. So, Q will have dimension $[d, d]$ to be compliant with the state and R will have dimension $[k, k]$ (practically a scalar), compliant to the input. P will be a $[d, d]$ square matrix. The goal of iLQR is to iteratively calculate K, the gain matrix that, multiplied by the error, returns the optimal policy, which in this case is encoded in a control signal. What we practically do is iterating backwards, starting from $t = t_final$, where we know the q-function is 0. This is basically like doing value iteration in closed form.

Online resources

- Sutton Barto
- https://rltheory.github.io/

Colleagues I consulted with

- Claudio Schiavella
- Andrea Marzo
- Nicoló Dentale
- Charlotte Primiceri