# Exercises: Tabular model-based algorithms

Statistical Planning and Reinforcement Learning

The purpose of these exercises is to help you understand tabular model-based reinforcement learning algorithms.

## 1 Practical exercises

Use your grid world environment model to test your implementation of the following exercises.

1. Implement the policy evaluation algorithm by following the interface suggested in Listing 1 and use this algorithm to evaluate a deterministic policy of your choice.

   The function *policy_evaluation* receives an environment model, a deterministic policy, a discount factor, a tolerance parameter, and a maximum number of iterations. A deterministic policy may be represented by an array that contains the action prescribed for each state.

   Listing 1: Policy evaluation interface.

```python
def policy_evaluation(env, policy, gamma, theta, max_iterations):
    value = np.zeros(env.n_states, dtype=np.float)

    # TODO: Implement policy evaluation

    return value
```

2. Implement the policy improvement algorithm by following the interface suggested in Listing 2.

   The function *policy_improvement* receives an environment model, a deterministic policy, the value function for this policy, and a discount factor.

   Listing 2: Policy improvement interface.

```python
def policy_improvement(env, policy, value, gamma):
    improved_policy = np.zeros(env.n_states, dtype=int)

    # TODO: Implement policy improvement

    return improved_policy
```

3. Implement policy iteration by combining your implementations of policy evaluation and policy improvement and by following the interface suggested in Listing 3.

   The function *policy_iteration* receives an environment model, a discount factor, a tolerance parameter, and a maximum number of iterations.

   Listing 3: Policy iteration interface.

```python
def policy_iteration(env, gamma, theta, max_iterations):
    policy = np.zeros(env.n_states, dtype=int)
    value = np.zeros(env.n_states, dtype=np.float)

    # TODO: Implement policy iteration

    return policy, value
```

4. Implement value iteration by following the interface suggested in Listing 4.

   The function *value_iteration* receives an environment model, a discount factor, a tolerance parameter, and a maximum number of iterations.

Listing 4: Value iteration interface.

```
def value_iteration(env, gamma, theta, max_iterations):
    policy = np.zeros(env.n_states, dtype=int)
    value = np.zeros(env.n_states, dtype=np.float)

    # TODO: Implement policy iteration

    return policy, value
```