HW #4: Value/Policy Iteration, Discretization

Name: YOUR NAME HERE

Deliverable: Link to the GitHub branch containing your solution.
Graduate Students: You are expected to complete the entire assignment.
Undergraduate Students: You need only complete questions that do not have **(GRAD)** next to them.

First, you will need to install matplotlib, numpy, scipy, and the Gymnasium Environment for Python: (if you have used gym before, gymnasium is the new supported version. It has some minor changes, but should feel mostly the same.)

```
pip install numpy
pip install gymnasium
pip install matplotlib
pip install scipy
```

or

```
conda install numpy
conda install gymnasium
conda install matplotlib
conda install scipy
```

Next, you need to install the HW4-RL python package.

Make sure you are in the correct directory.

```
cd /path/to/your/code/HW4-RL
```

Example:

```
cd /home/brendan/advanced_robotics/homework/HW4-RL
```

Install the package using pip

```
pip install -e .
```

or

```
pip install .
```

Now you should be good to go!

For more install and usage instructions see README.md

1 Gridworld Env

1.1 Value Iteration [20pts]

First, you will implement the value iteration algorithm for the tabular case. You must fill the code in `code/tabular_solution.py` below the lines `if self.policy_type == 'deterministic_vi'`. Run the script for the two gridworld domains and report the heatmap of the converged values.

1.2 Policy Iteration [20 points]

Next, you will implement the policy iteration algorithm for the tabular case. You will need to fill the code in `code/tabular_solution.py` below the lines `if self.policy_type == 'deterministic_pi'`. Run the script for the first gridworld domain and turn in a graph with policy value (accumulated reward) on the vertical axis and iteration number on the horizontal axis.

2 Mountain Car

2.1 Near neighbors iterpolation [20pt]

Value Iteration can only work when the state and action spaces are discrete and finite. If these assumptions do not hold, we can approximate the problem domain by coming up with a discretization such that the previous algorithm is still valid. The *MountainCar* domain, as described in class, has a continuous state space that will prevent us from using value or policy iteration to solve it directly. One of the solutions that we came up with for this was *nearest-neighbor interpolation*, where we discretize the actual state space S into finitely many states $\Xi = \xi_1, \xi_2, ..., \xi_n$, and act as if we are in the ξ nearest to our actual state s.

Implement Value Iteration with nearest-neighbor interpolation on the *MountainCar* domain. You will need to add code in *code/continuous_solution.py* below the lines *if self._mode == 'nn'*. Run the script and report the state value heatmap for `MountainCar`, discretizing each dimension of the state space (position and velocity) into 21, 51, and 101 bins.

2.2 Linear interpolation

Nearest-neighbor interpolation is able to approach the optimal solution if you use a fine-grained approximation, but doesn't scale well as the dimensionality of your problem increases. A more powerful discretization scheme that we discussed in class is *n-linear interpolation*, an n-dimensional analogue of linear interpolation. Add your code within `code/continuous_solution.py` below the line `if self._mode == 'linear'`. Just as before, report the state value heatmap for `MountainCar` with discretization resolutions of 21, 51, and 101 points per dimension.

2.3 Stochastic Policy Iteration - Grad Only

Grad students, implement stochastic policy iteration for the Mountian Car problem. This should be a small update to your existing policy iteration implementation.

a) You should not be considered if your stochastic policy iteration does not work as well as your deterministic implementation, this is expected. Why might this be? Explain your hypothesis.

b) The idea of a stationary policy is more difficult to determine in the stochastic case. We have provided an implementation of KL-divergence for you in the codebase, you may use this to determine if your policy is approximately stationary. What is KL-divergence, and why might it be a good way of determining if your policy is stable? If you chose to use something other than KL-divergence, briefly explain.