Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for Dynamic Systems and Control
IDSC
Institut für Dynamische Systeme und Regelungstechnik

151-0563-01 **Dynamic Programming and Optimal Control** (Fall 2019)

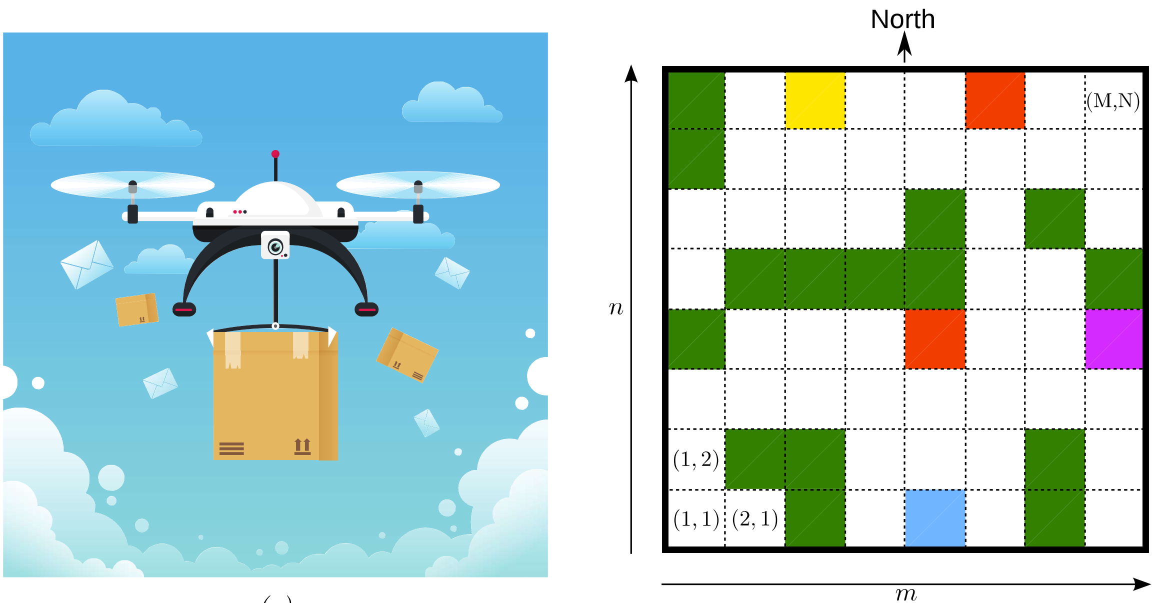
Programming Exercise Topic: Infinite Horizon Problems

Issued: Nov 20, 2019 Due: Dec 20, 2019

Nikita Rudin(rudinn@ethz.ch), David Hoeller(dhoeller@ethz.ch), November 20, 2019

Policy Iteration, Value Iteration, and Linear Programming

The goal of this programming exercise is to deliver a package with a drone as quickly as possible. To achieve this, the drone must first fly to a pick-up station to collect a package and then reach a delivery station to discharge it. Along the way, the drone must avoid hazards such as trees or angry residents who try to shoot it down.



(a) (b)

Figure 1: (a) Drone delivering a package (image source: Freepik.com). (b) Top-down view of an example world. The purple cell represents the pick-up station, the yellow cell the base station, the blue cell the delivery station, the green cells the trees, and the red cells the angry residents.

Problem set up

The drone operates in a world that is discretized into $M \times N$ cells (Fig. 1b), where M is the width of the world (from west to east) and N is the length (from south to north). The state of the drone at time step k is described by $x_k = (m, n, \psi)$, where $m \in \{1, \dots, M\}$ and $n \in \{1, \dots, N\}$ describe the position of the drone along the west to east and south to north axes, respectively, and $\psi \in \{0, 1\}$, where 1 indicates that the drone is carrying a package and 0 indicates that it has no package.
At time step k, the system evolves in the following way:

1. One of the allowable control inputs u_k is applied. The drone is able to move north, west, south, or east by one cell or hover (stay). However, if an input would move the drone to a cell with a tree or reach out of the bounds of the world, that input is not allowed.

2. From the new position, a gust of wind can occur with probability p_{wind}. This moves the drone either north, west, south, or east, all with equal likelihood.

3. If the drone ends up in a cell with a tree or leaves the bounds of the world, the drone crashes.

4. If the drone has not crashed by this point, the angry residents will attempt to shoot it down. The probability of being hit by a resident i is

$$p_{r_i} = \begin{cases} \gamma/(d_i + 1) & \text{if } 0 \leq d_i \leq R \\ 0 & \text{otherwise} \end{cases},$$

   where d_i is the L_1 distance (measured in number of cells) from the drone to resident i, and R is the maximum shooting range of the residents. All residents shoot simultaneously so that the drone can be hit multiple times. Being hit results in a crash.

5. If the drone has not crashed by this point and is at a pick-up station carrying no package, it collects one. If the drone has not crashed by this point and is at a delivery station carrying a package, the task terminates.

Whenever the drone crashes, it is brought to the base station and starts the next time step there without a package. This procedure takes a total of N_c time steps.

Note 1: The events take place in this exact order, 1 to 5.
Note 2: The drone does not collide with residents and can thus find itself in a cell with a resident without crashing. Trees do not affect the residents' shooting capabilities.

Tasks

Find the policy minimizing the expected number of time steps required to successfully deliver a package by

a) finding the index of the terminal state in the state space matrix. As you will see in `main.m`, the state space matrix has K rows, where each row corresponds to a possible value x can take.
Use the `ComputeTerminalStateIndex.m` file provided as a template for your implementation.

b) creating a transition probability matrix $P \in \mathbb{R}^{K \times K \times L}$, where K is the number of possible states and L is the number of control inputs. To compute P, each entry in the state space is assigned a unique index $i = 1, 2, \ldots, K$.
Use the `ComputeTransitionProbabilities.m` file provided as a template for your implementation.
This part counts 30% towards the grade.

c) creating a stage cost matrix $G \in \mathbb{R}^{K \times L}$.
Use the `ComputeStageCosts.m` file provided as a template for your implementation.
This part counts 25% towards the grade.

d) applying Value Iteration[1], Policy Iteration and Linear Programming to compute $J \in \mathbb{R}^K$ and the optimal policy $\mu(i),\ i = 1, \ldots, K$, that solves the stochastic shortest path problem.
Use the `ValueIteration.m`, `PolicyIteration.m`
and `LinearProgramming.m` files provided as a template for your implementation.
Each algorithm contributes 15% of the grade.

[1] You can terminate the algorithm if all $J(i),\ i = 1, \ldots, K$, do not change by more than 10^{-5} within one iteration step.

Matlab files provided

A set of MATLAB files is provided on the class website. Use them to solve the above problem. Follow the structure strictly as grading is automated.

`main.m`	Matlab script that has to be used to generate the world, execute the stochastic shortest path algorithms and display the results.
`GenerateWorld.p`	Matlab function that generates a random world.
`MakePlots.p`	Matlab function that can plot a map of the world, and the cost and control action for each accessible cell.
`ComputeTerminalStateIndex.m`	Matlab function template to be used to compute the index of the terminal state in the state space matrix.
`ComputeTransitionProbabilities.m`	Matlab function template to be used for creating the transition probability matrix $P \in \mathbb{R}^{K \times K \times L}$.
`ComputeStageCosts.m`	Matlab function template to be used for creating the stage cost matrix $G \in \mathbb{R}^{K \times L}$.
`ValueIteration.m`	Matlab function template to be used for your implementation of the Value Iteration algorithm for the stochastic shortest path problem.
`PolicyIteration.m`	Matlab function template to be used for your implementation of the Policy Iteration algorithm for the stochastic shortest path problem.
`LinearProgramming.m`	Matlab function template to be used for your implementation of the Linear Programming algorithm for the stochastic shortest path problem.
`exampleWorld.mat`	A pre-generated world to be used for testing your implementations of the above functions.
`exampleP.mat`	The transition probability matrix $P \in \mathbb{R}^{K \times K \times L}$ for the example world.
`exampleG.mat`	The stage cost matrix $G \in \mathbb{R}^{K \times L}$ for the example world.

Deliverables

Please hand in by e-mail

- your MATLAB implementation of the following files:
  `ComputeTerminalStateIndex.m`,
  `ComputeTransitionProbabilites.m`,
  `ComputeStageCost.m`,
  `ValueIteration.m`,
  `PolicyIteration.m`,
  `LinearProgramming.m`.
  Only submit the above mentioned files. Your code should not depend on any other non-standard MATLAB functions.

- A scanned declaration of originality in a `PDF`-file, signed by each student to confirm that the work is original and has been done by the author(s) independently:
  `https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/declaration-originality.pdf`.

Up to three students are allowed to work together in a team and need to submit only once. They will all receive the same grade.
Please include all files into one `zip`-archive, named `DPOCEx_Names.zip`, where `Names` is a list of the full names of all students who have worked on the solution.
(e.g `DPOCEx_RudinNikita_HoellerDavid.zip`)

Send your files to `dhoeller@ethz.ch` with subject `[programming exercise submission]` by the due date indicated above. We will send a confirmation e-mail upon receiving your e-mail. You are ultimately responsible that we receive your solution in time.
Important: Each work submitted will be tested for plagiarism.