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 2017)

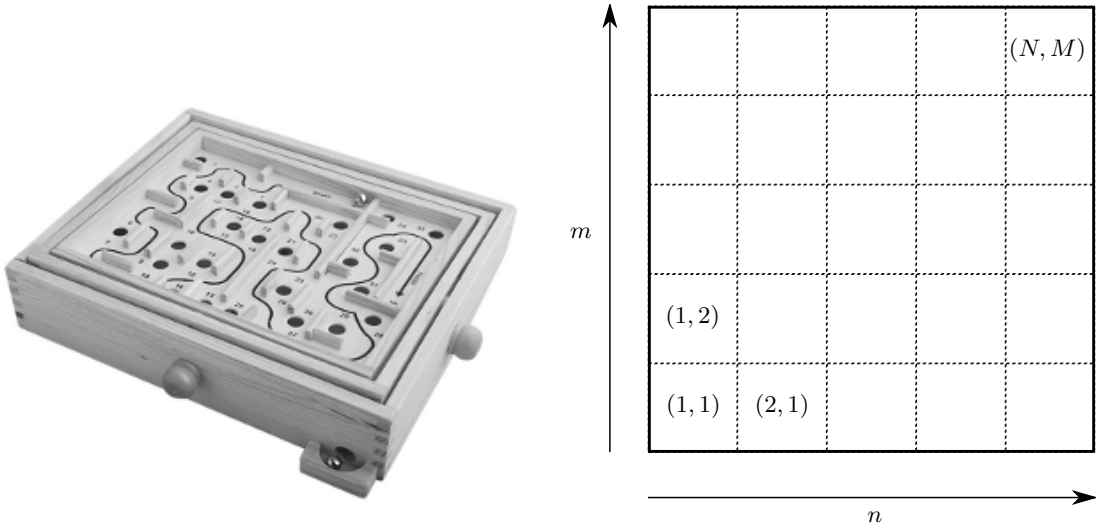
Programming Exercise Topic: Infinite Horizon Problems

Issued: Nov 22, 2017 Due: 23:59:59, Dec 19, 2017

Rajan Gill(rgill@ethz.ch), Weixuan Zhang(wzhang@ethz.ch), November 22, 2017

Policy Iteration, Value Iteration, and Linear Programming

The goal of this programming exercise is to find a policy which minimizes the number of expected time steps to guide a ball through a maze to a given target cell.



(a) Picture of a ball-and-roller maze. *(b) Grid used to discretize the maze.*

Figure 1

The maze is discretized into $N \times M$ cells (see Fig. 1b), where N is the width of the maze and M the height, respectively. The ball state is described by its position $x_k = (n, m)$, $n \in \{1, \dots, N\}$, $m \in \{1, \dots, M\}$. At each time step, the ball position can be controlled by tilting the maze and is subject to a disturbance. If there are no walls present, the ball position evolves as follows:

1. First, the control input $u_k \in \left\{u \in \mathbb{Z}^2 |\ u = se,\ s \in \{0, 1, 2\},\ \|e\|_\infty = 1\right\}$ is applied and the ball moves to position $x_k + u_k$;

2. Then, a disturbance w_k occurs and the ball is moved to $x_k + u_k + w_k$,
where $w_k \in \{w \in \mathbb{Z}^2 \mid \|w\|_\infty \leq 1\}$ with all elements having equal probability.

If there are walls present, the control space is adapted accordingly to only allow for feasible movements. Diagonal movements of the ball are only allowed if no wall ends at or passes through the connecting corner. After the control input u_k is applied, the disturbance w_k acts on the ball position using the same uniform probability distribution described above. However, if the ball would cross a wall due to the disturbance, it bounces off the wall and reenters the current cell.

If the ball ends up in the target cell after the control input and disturbance have been applied, it remains there for all future time steps.

The maze also contains holes. When traversing through or to a cell with a hole, there is a probability p_f of falling into it. If the ball falls into the hole, it is moved to a reset cell at the beginning of the next time step.

Time penalties will be incurred if the ball bounces into a wall or falls into a hole. If the ball falls into a hole, an additional time penalty of c_r time steps occurs. If the ball hits a wall or the boundary after the disturbance is applied, an additional time penalty of c_p time steps is incurred.

The K wall segments of the maze are provided by a $2K \times 2$-matrix, where the start and end point of the k-th wall segment are stored in row $2k-1$ and $2k$, respectively.

Tasks

Find the policy minimizing the number of time steps required to travel to the target cell by

a) creating a transition probability matrix $P \in \mathbb{R}^{MN \times MN \times L}$, where L is the maximum number of attainable control inputs[1]. For creating P you need to label all cells of the maze with a unique node number $i = 1, 2, \ldots, MN$. For example cell $x = (1, 1)$ becomes node $i = 1$, $x = (1, 2)$ becomes $i = 2$, and $x = (2, 1)$ becomes $i = M + 1$.
Use the provided file `ComputeTransitionProbabilities.m` as a template for your implementation.
This part counts 30% towards the grade.

b) creating an expected stage cost matrix[2] $G \in \mathbb{R}^{MN \times L}$.
Use the provided file `ComputeStageCosts.m` as a template for your implementation.
This part counts 25% towards the grade.

c) applying Value Iteration, Policy Iteration and Linear Programming[3] to compute $J \in \mathbb{R}^{MN}$ and the optimal policy $\mu(i) \in \mathcal{U}(i)$, $i = 1, \ldots, MN$, that solves the stochastic shortest path problem.
Use the provided files `ValueIteration.m`, `PolicyIteration.m` and `LinearProgramming.m` as a template for your implementation.
Each algorithm makes up for 15% of the grade.

Note: The matrices P, G, J and the vector μ also contain values for the target cell in order to simplify state indexing.

[1] Set the transition probabilities to 0 for infeasible moves.

[2] Set the expected stage cost to `inf` for infeasible moves.

[3] In your implementation of the file LinearProgramming.m, you may use the MATLAB function "linprog" to solve the linear program.

Provided Matlab files

A set of MATLAB files is provided on the class website. Use them for solving the above problem[4].

`main.m`	Matlab script that can be used to generate a maze, execute the stochastic shortest path algorithms and display the results.
`GenerateMaze.p`	Matlab function that generates a maze with walls, target cell and optionally holes.
`PlotMaze.m`	Matlab function that can plot a maze, the costs for each cell and the control action in each cell.
`ComputeTransitionProbabilities.m`	Matlab function template to be used for creating the transition probability matrix $P \in \mathbb{R}^{MN \times MN \times L}$.
`ComputeStageCosts.m`	Matlab function template to be used for creating the expected stage cost matrix $G \in \mathbb{R}^{MN \times L}$.
`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.
`pregeneratedMaze.mat`	A pregenerated 6×5 maze to be used for testing your implementations of the above functions.

[4]Strictly follow the structure. Grading is automated. You can add functions to the template files, but each file should be self-contained, i.e. not depend on any external custom function.

Deliverables

Please hand in by e-mail

- your MATLAB implementation of the following files:
  `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 (you can find the PDF file in the template). It should be signed by each student to confirm that the work is original and has been done by the author(s) independently. Each work submitted will be tested for plagiarism.

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 `DPOCEx2_RajanGill_WeixuanZhang.zip`)[5]

Send your files to `wzhang@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.

[5] Up to three students are allowed to work together on the problem. They will all receive the same grade.