151-0563-01 **Dynamic Programming and Optimal Control** (Fall 2014)

Programming Exercise Topic: Infinite Horizon Problems

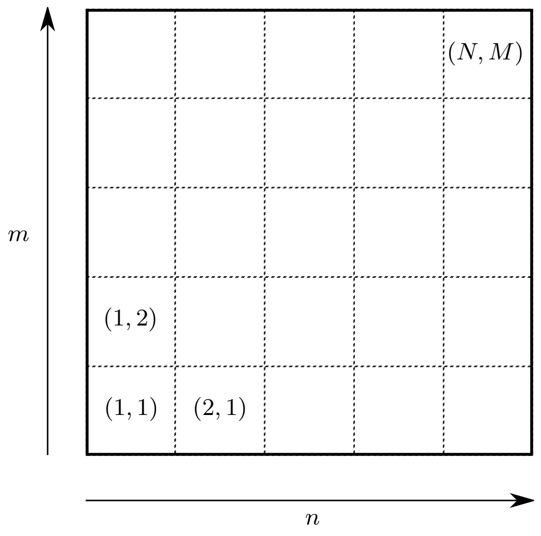
Issued: Nov 19, 2014 Due: Dec 16, 2014

Dario Brescianini(bdario@ethz.ch), Robin Ritz(rritz@ethz.ch), November 19, 2014

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.

(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 $x = (n, m)$, $n \in \{1, \ldots, N\}$, $m \in \{1, \ldots, M\}$. For 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

$$x_{k+1} = x_k + u_k + w_k, \qquad k = 0, 1, \ldots$$
$$u_k \in U(x_k) = \{u \in \mathbb{Z}^2 \mid \|u\|_1 \leq 2\},$$
$$w_k \in W(x_k, u_k) = \{w \in \mathbb{Z}^2 \mid \|w\|_1 \leq 1\}, \text{ with all elements having equal probability.}$$

If two cells are separated by a wall, the control space $U(x_k)$ is adapted accordingly to only allow 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, a disturbance w_k acts on the ball position. If the ball would cross a wall due to the disturbance, it remains at its position. 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 K wall segments of the maze are provided by a $2 \times 2K$-matrix, where the start and end point of the k-th wall segment are stored in column $2k - 1$ and $2k$, respectively.

Tasks - Part I

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. 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 `ComputeTransitionProbabilitiesI.m` as a template for your implementation.
This part counts 20% towards the grade.

b) creating a stage cost matrix $G \in \mathbb{R}^{MN \times L}$.
Use the provided file `ComputeStageCostsI.m` as a template for your implementation.
This part counts 5% towards the grade.

c) applying Value Iteration, Policy Iteration and Linear Programming to compute $J \in \mathbb{R}^{MN}$ and the optimal policy $\mu(i) \in U(i)$, $i = 1, \ldots, MN$, 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.

Tasks - Part II

A more accurate model of the game in Fig. 1a also contains holes and penalties if the ball bounces into a wall. If the ball falls into a hole after the control input u_k **OR** after the disturbance w_k has been applied, the ball is moved to a reset cell at the beginning of the next time step with an additional time penalty of c_r time steps. It is not possible to jump over a hole. If the ball bounces into a wall, a time penalty of c_p time steps is incurred.
Find the policy minimizing the number of time steps required to travel to the target cell by

d) adapting the transition probability matrix P to model the dynamics of the holes.
Use the provided file `ComputeTransitionProbabilitiesII.m` as a template for your implementation.
This part counts 15% towards the grade.

e) adapting the stage cost matrix G to account for the time penalty c_p if the ball bounces into a wall or c_r if the ball falls into a hole and has to restart at the reset cell.
Use the provided file `ComputeStageCostsII.m` as a template for your implementation.
This part counts 5% towards the grade.

Readability of the submitted code counts 10% towards the grade.

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

Provided Matlab files

A set of MATLAB files is provided on the class website. Use them for solving the above problem[1].

`ScriptI.m` `ScriptII.m`	Matlab script that can be used to generate a maze, execute the stochastic shortest path algorithms and display the results of part I and part II, respectively.
`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.
`ComputeTransitionProbabilitiesI.m` `ComputeTransitionProbabilitiesII.m`	Matlab function template to be used for creating the transition probability matrix $P \in \mathbb{R}^{MN \times MN \times L}$ of part I and part II, respectively.
`ComputeStageCostsI.m` `ComputeStageCostsII.m`	Matlab function template to be used for creating the stage cost matrix $G \in \mathbb{R}^{MN \times L}$ of part I and part II, respectively.
`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.
`controlSpace.mat`	A $L \times 2$-matrix, where each row contains an element of the control space.
`disturbanceSpace.mat`	A $S \times 3$-matrix, where the first two columns contain the S elements of the disturbance space and the third column represents the corresponding probability.
`pregeneratedMazeI.mat` `pregeneratedMazeII.mat`	A pregenerated 10×10 maze to be used for testing your implementations of the above functions for part I and part II, respectively.

[1]Strictly follow the structure. Grading is automated.

Deliverables

Please hand in by e-mail

- your MATLAB implementation of the following files:
  `ComputeTransitionProbabilitesI.m`,
  `ComputeTransitionProbabilitesII.m`,
  `ComputeStageCostI.m`,
  `ComputeStageCostII.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.

- in a `PDF`-file a scanned declaration of originality, signed by each student to confirm that the work is original and has been done by the author(s) independently:
  http://www.idsc.ethz.ch/Courses/optimal_control/declaration_of_originality.pdf.
  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_BrescianiniDario_RitzRobin.zip`)[2]

Send your files to `bdario@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.

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