Assignment 2:
Design strategies for iterated prisoner's dilemma

ENM140, Game theory and rationality 2016

1 Overview

With this assignment, we organize a competition where the course participants contribute strategies for playing the iterated prisoners dilemma. In the competition each strategy will play against all other strategies (but not against itself). Your job is to implement the strategies as functions in Mathematica. We have written all the code for running the game, so you only have to write a set of functions that return a value corresponding to "defect" or "cooperate".

We will play three versions of the game:

1. Played for 10 rounds. Submit 3 strategies for this version.
2. Played for 200 rounds. Submit 3 strategies for this version.
3. Played for 200 rounds with a mistake rate of 2%. This means that your chosen action will be used with 98% probability and otherwise changed to the other action. Submit 1 strategy for this version.

We will run the game with payoffs $u_{dd} = 1, u_{dc} = 5, u_{cd} = 0, u_{cc} = 3$, where u_{dd} is the payoff for player 1 if both defect, u_{dc} is the payoff if player 1 defects and player 2 cooperates, etc. The winning strategy in each version of the game will the one who gets the highest average payoff across all the matches against other strategies.

Please read these instructions carefully before starting your work.

2 How to get started

First of all, download a zip file with Mathematica code from the course homepage. Unpack the zip file and you will have a folder called `PDTournament2016` with a few different files in it:

- The notebook `PD-tournament.nb` is the best place to start developing your own strategies. More about this in a moment.
- The file `GameTournament.m` contains all code for actually running the game. You don't have to read this unless you want to.

- The notebook `TestStrategies.nb` and the file `Strategies/examplecid.m` are for testing and submitting your work. More about this in Section 3.

2.1 Develop your strategies in `PD-tournament.nb`

Start Mathematica and open the notebook `PD-tournament.nb`. Run all the cells, e.g. by clicking `Evaluation > Evaluate Notebook` in the menu. When the notebook has finished running, read through the code and the output to get a feeling for how things work.

Note the section where there are strategies named like `examplecid`N`variant`, where $N \in \{10, 200\}$ and $variant \in \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{mistakes}\}$. This is where you should insert your own strategies. The numbers indicate how many rounds will be played with the strategy and the letters `A`, `B`, `C` identify your three different strategies in the first two variants of the game. The strategy ending in `200mistakes` is the one that will be used in the game with 2% mistake rate.

Using `history` in your strategies

Look closely at the example strategies (e.g. "always cooperate", "always defect", "tit-for-tat", etc). Note that they are all functions that take a single argument, the `history`, and always return either 0 (= defect) or 1 (= cooperate). The `history` passed to the functions will always be a list of the form

$$\text{history} = \{\{m_1, o_1\}, \{m_2, o_2\}, \ldots, \{m_{t-1}, o_{t-1}\}\},$$

where m_i is "my" action at time i and o_i is the opponent's action. So for example, a tit-for-tat strategy can be mathematically defined as

$$\begin{cases} m_1 = 1, \\ m_t = o_{t-1}, \quad t = 2, 3, \ldots. \end{cases}$$

At time $t = 1$, the history is empty, i.e. history $= \{\}$, and then the tit-for-tat strategy generously starts with cooperation, $m_1 = 1$. In all following time steps, the tit-for-tat strategy always copies what the opponent did last.

In Mathematica, the tit-for-tat strategy can be formulated as follows:

```
titForTat[history_] := If[
   Length[history] == 0,
   1, (* Cooperate initially *)
   Last[Last[history]] (* Then copy the opponent *)
   ];
```

The current round number t is equal to `Length[history] + 1`. Note that if you want to use history, you can only use it at time $t = 2$ and later. Use something like the `If`-clause above to check that `Length[history] > 0`.

A few specific hints for how to use the history:

```
(* the last pair of actions, i.e. {m_{t-1}, o_{t-1}} *)
Last[history]

(* the opponent's action in the previous round *)
Last[Last[history]]

(* your action in the previous round *)
First[Last[history]]

(* all your previous actions, then all the opponent's actions *)
Transpose[history]

(* all your previous actions, i.e. {m_1, m_2, ..., m_{t-1}} *)
First[Transpose[history]]

(* all the opponent's previous actions, i.e. {o_1, o_2, ..., o_{t-1}} *)
Last[Transpose[history]]
```

Using state variables

For some strategies you might want to keep track of some state in your strategy. This can be inconvenient or impossible to do without some additional variable. So we allow you to use additional state variables for your strategies, but you have to name them in a special way. For the strategy *Nvariant*, you may only use state variable names containing `state`*Nvariant*, for example:

```
state10Afirst
state10Asecond
state200C
```

All state variables named like this will automatically be deleted before each game, and then a specially named intialization function will be run for you. The initialization function should look something like this:

```
examplecidInitState := Module[
   {},
   state10Afirst = "something";
   state10Asecond = 42;
   state200C = False;
   (* and so on, as needed *)
   ];
```

There is an example in the notebook that may help to explain this:

```
examplecidInitState := Module[
   {},
   state200C = False;
   ];

examplecid200C[history_] := Module[
   {turnedEvil = state200C, turnEvilProbability = 0.1},
   state200C = turnedEvil || (RandomReal[] < turnEvilProbability);
   Return[If[turnedEvil, 0, 1]]; (* Defect iff evil *)
];
```

What it means is the following: Strategy `200C` is to cooperate unconditionally, unless the agent suddenly turns madly evil. The state variable `state200C` is initialized as `False`. In

each round the state variable `state200C` is set to `True` if it is already `True` or if a random number $r \sim \text{Uniform}(0, 1) < 0.1$, otherwise it is kept `False`. In other words, once the strategy goes evil it will never go back. The strategy returns 0 (defect) if evilness has struck and 1 (cooperate) otherwise.

3 How to submit

Work on your strategies as much or little as you want. You may use the same strategy in several cases, but all the seven functions (10A, 10B, 10C, 200A, 200B, 200C, 200mistake) must be defined.

When you are done, follow these instructions to package the files in a separate file, test them, and submit them:

1. Make a copy of the file `Strategies/examplecid.m` and name it `Strategies/`*`yourcid`*`.m` where *`yourcid`* is your own CID. You can do this by opening the file `examplecid.m` and using `File > Save As...`. Please make sure you save the file as a **Wolfram Mathematica Package *.m** file, not a notebook!

2. Open the `Strategies/`*`yourcid`*`.m` and replace the placeholder contents by your own strategy functions and your own `InitState` function in there. It should work perfectly well to copy that whole block of code from the `PD-tournament.nb` notebook file where you developed the strategies.

3. Replace all occurrences of `examplecid` by *`yourcid`* in the `Strategies/`*`yourcid`*`.m` file. We recommend that you use the Find and replace tool (`Edit > Find...` in the menu) to do this.

4. Save the `Strategies/`*`yourcid`*`.m` file.

5. Open the Mathematica notebook called `TestStrategies.nb`.

6. At the top of the notebook, change the line that says
   `cid = "examplecid";` to `cid = "`*`yourcid`*`";`

7. Restart the Mathematica kernel by clicking `Evaluation > Quit Kernel > Local` in the menu. This resets all variables that might be set and possibly affect the testing code.

8. Run the TestStrategies notebook, e.g. by clicking `Evaluation > Evaluate Notebook` in the menu. It will play a short repeated game against a random opponent and check that the output looks reasonable. You should see output like `Testing case 10, A OK!`, for all the 7 strategies. If there is a problem, you will see a more or less helpful error message. If there is a problem, try to fix it and then reset the Mathematica kernel again before running the whole TestStrategies notebook again.

9. When the testing notebook says `OK!` to everything, you are done. Go to Assignment 2 on Ping Pong and submit your file called `Strategies/`*`yourcid`*`.m`.