ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

Department of Electrical
and Computer Engineering

Game Theory

Evolutionary Games Toolbox - Documentation

Team	2
Vaporis Dimitrios	AEM 10625
Voulkidis Vlasios	AEM 10627
Delopoulos Emmanouil	AEM 10693
Professor	Kehagias Athanasios

Spring Semester 2025

Contents

1 Introduction

The following is a file containing the documentation for the Evolutionary Games Toolbox contained in this GitHub repository. It is split into three segments: in the first segment, the documentation of each function contained in the Code folder are presented, whereas in the second segment, the simulation correspoding to each example file in the Examples folder is noted. Lastly, the third segment contains documentation on the strategies in the strategies subfolder of the Code folder. Note that the details regarding the operation of each function are not presented in this file (a lot of this is done in the Report.pdf file in the Report folder). Instead, this is a showcase of what each function expects as input, what it returns as output, which example corresponds to which simulation and how each strategy of the toolbox behaves. Across the functions written, there are smaller helper functions that are not included in this documentation, for the sake of simplicity.

2 Functions of Code folder

This is the documentation of the functions contained in the Code folder. The inputs/outputs of each functions are presented, without showcasing the operation of each one. See also the Report.pdf as well as the source code of each function (it is properly commented) if necessary.

2.1 AnalyzeMarkovChain

The AnalyzeMarkovChain function is responsible for creating the directed state graphs of the Imitation Dynamics. It requires inputs P the transition matrix of the Markov Chain (usually calculated by the TourTheImi function), $POP0$ the initial $1 \times S$ (with S the number of strategies used in the simulation) population vector, *Strategies* an array containing the string names of the strategies used in the simulation (as named later on in the Strategies section) and *Title* the string containing the title of the outgoing graph. The function does not have any outputs, as it only creates the graph and exports it for use in the Report).

2.2 Axel

The Axel function computes the score of a single generation Axelrod tournament between specific strategies. It expects inputs B a 2×2 payoff matrix for each match, *Strategies* an array containing the string names of the strategies used in the simulation (as named later on in the Strategies section), *Pop* the initial $1 \times S$ (with S the number of strategies used in the simulation) population vector and T the number of rounds played in each match of the tournament.

It returns an $N \times 1$ array (with N being the total number of players in the simulation) containing the score of each player of the simulation at the end of the Axelrod tournament. Note that this function is not used for functions later on because of other faster implementations used.

2.3 plotFitnessVSImitation

The plotFitnessVSImitation function creates and exports a figure comparing the Fitness and Imitation Dynamics, using both "Total" and "Individual" mode for best strategy calculation in the Imitation Dynamics. It expects inputs *Strategies* an array containing the string names of the strategies used in the simulation (as named later on in the Strategies section), *POP*1 the population matrix for each generation of the Fitness Dynamics (created by TourSimFit), *POP*2 the population matrix for each generation of the Imitation Dynamics - "Individual" mode (created by TourSimImi), *POP*3 the population matrix for each generation of the Imitation Dynamics - "Total" mode (created by TourSimImi with "Total" mode input) and *Title* the title string of the created graph. The function does not have outputs and only creates and exports the graph. See example10 for an example usage of the function.

2.4 plotPopulationOfStrategiesOverGenerations

The plotPopulationOfStrategiesOverGenerations function constructs the diagram of populations of different strategies across generations after a completed simulation. It requires inputs *Strategies* an array containing the string names of the strategies used in the simulation (as named later on in the Strategies section), *POP* the population matrix for each generation of the given simulation, created by the corresponding function and *Title* the title string of the created diagram. The function does not have outputs and only creates and exports the diagram graphing the population of each strategy with respect to the generation number.

2.5 plotPopulationsOfStrategiesOverGenerations

The plotPopulationsOfStrategiesOverGenerations function creates the plots for the population of different strategies across generations for results generated by TourTheFit, TourSimFit with compensation and without compensation. It expects inputs *Strategies* an array containing the string names of the strategies used in the simulation (as named later on in the Strategies section), *POP*1 the theoretical population matrix for each generation of the Fitness Dynamics (created by TourTheFit), *POP*2 the population matrix for each generation of the Fitness Dynamics without compensation (created by TourSimFit), *POP*3

the population matrix for each generation of the Fitness Dynamics with compensation (created by TourSimFit with *compensation* = true input) and *Title* the title string of the created figure. The function does not have outputs and only creates and exports the figure. See the first few examples for example usages of the function.

2.6 plotPopulationsTourSimImi

The plotPopulationsTourSimImi function creates the plot for the population of different strategies across generations after a completed TourSimImi simulation. It expects inputs *Strategies* an array containing the string names of the strategies used in the simulation (as named later on in the Strategies section), *POP* the population matrix for each generation of the given simulation, created by the TourSimImi function and *Title* the title string of the created plot. The function does not have outputs and only creates and exports the diagram graphing the population of each strategy with respect to the generation number. See example07 for an example usage of the function.

2.7 TourSimFit

The TourSimFit function simulates the evolutionary tournament using Fitness Dynamics. The function expects inputs B a 2×2 payoff matrix for each match, *Strategies* an array containing the string names of the strategies used in the simulation (as named later on in the Strategies section), $POP0$ the initial $1 \times S$ (with S the number of strategies used in the simulation) population vector, T the number of rounds played in each match of the tournament, J the number of generations of the simulation and *compensation* an optional argument (with default value being false) that toggles the usage of random compensation to keep the total population of each generation constant (this is better explained in the Report). It returns a $(J+1) \times S$ array *POP* containing the population of each generation for each strategy, a $J \times S$ array *BST* containing information about the best strategy of each generation (for the cell (i,j) of the array, if it has value 1 then strategy j was the best for generation i, else it has value 0) and *FIT* a $J \times S$ array that contains the fitness scores of each strategy for each generation.

2.8 TourSimImi

The TourSimImi function simulates the evolutionary tournament using Imitation Dynamics. The function expects inputs B a 2×2 payoff matrix for each match, *Strategies* an array containing the string names of the strategies used in the simulation (as named later on in the Strategies section), $POP0$ the initial

$1 \times S$ (with S the number of strategies used in the simulation) population vector, T the number of rounds played in each match of the tournament, J the number of generations of the simulation and *mode* an optional argument (with default value being "Individual") that toggles the mode for best strategy selection, with the other option being "Total" (this is better explained in the Report). It returns a $(J+1) \times S$ array *POP* containing the population of each generation for each strategy and a $J \times S$ array *BST* containing information about the best strategy of each generation (for the cell (i,j) of the array, if it has value 1 then strategy j was the best for generation i, else it has value 0).

2.9 TourTheFit

The TourTheFit function conducts the theoretical analysis of an evolutionary tournament using Fitness Dynamics. The function expects inputs B a 2×2 payoff matrix for each match, *Strategies* an array containing the string names of the strategies used in the simulation (as named later on in the Strategies section), *POP*0 the initial $1 \times S$ (with S the number of strategies used in the simulation) population vector, T the number of rounds played in each match of the tournament and J the number of generations of the simulation. It returns a $(J+1) \times S$ array *POP* containing the population of each generation for each strategy, a $J \times S$ array *BST* containing information about the best strategy of each generation (for the cell (i,j) of the array, if it has value 1 then strategy j was the best for generation i, else it has value 0) and *FIT* a $J \times S$ array that contains the fitness scores of each strategy for each generation.

2.10 TourTheImi

The function TourTheImi computes the state transition matrix of a given evolutionary tournament using Imitation Dynamics using Markov Chain theory. The function expects inputs B a 2×2 payoff matrix for each match, *Strategies* an array containing the string names of the strategies used in the simulation (as named later on in the Strategies section), *POP*0 the initial $1 \times S$ (with S the number of strategies used in the simulation) population vector, T the number of rounds played in each match of the tournament, J the number of generations of the simulation and *mode* an optional argument (with default value being "Individual") that toggles the mode for best strategy selection, with the other option being "Total" (this is better explained in the Report). It returns an array P which is the state transition matrix of the process. More specifically, the cell (i,j) of the array contains the probability that the system reaches state j at the next step, given that the system starts at state i (with system being the population distribution and each state being each possible population distribution). Use AnalyzeMarkovChain to visualize and understand the

results.

3 Examples of Examples folder

This is the documentation of the Examples contained in the Examples folder of the repository. More specifically, the situation each example simulates is presented and any possible links to the paper by Mathieu et al are mentioned.

3.1 example01

3.2 example02

3.3 example03

3.4 example04

3.5 example05

3.6 example06

3.7 example07

3.8 example08

3.9 example09

3.10 example10

4 Strategies of strategies subfolder in the Code folder

Lastly, this is the documentation for all strategies included in the strategies subfolder of the Code folder. For more details, the exact source code is available for each of the following strategies. Note that you can always enhance this collection with strategies of your own. For your created strategies to work with the toolbox, follow the format used in the strategies given: let the function have form function Move = YourStrategy(History), where History is the $T \times 2$ matrix of moves by both players for each turn, regard the player of your strategy as player 1 (the first column of the History matrix, it is flipped in case the player is actually player 2) and make Move = 2 in case of defection and Move = 1 in case of cooperation. Also, suppose that each strategy can only access past moves (player 2 cannot access player 1's current round move) and that each strategy does not know the amount of rounds played in each match. Lastly, because of the current form of simulation and theoretical analysis functions

(that prioritize low running times), strategies including random elements are not included (in case you wish to include some, most functions will probably fail).

4.1 All_C

4.2 All_D

4.3 Alternator

4.4 Cycler

4.5 Detective

4.6 Grim

4.7 per_ccccd

4.8 per_ccd

4.9 per_ddc

4.10 Prober

4.11 SneakyTitForTat

4.12 soft_majo

4.13 SpitefulTitForTat

4.14 TitForTat

4.15 TitForTwoTats

4.16 TwoTitsForTat