Experience-Weighted Attraction Model
Python Implementation

Notebook Author:
Vinícius Ferraz

Affiliation:
Alfred Weber Institute of Economics, University of Heidelberg
Contact:
Correspondence should be addressed to visferraz@gmail.com

Version: 1.0.0

1. Introduction

This document is supposed to be used in conjunction of the EWA Jupyter Notebook implementation. The notebook contains all information about the theory behind the model, as well as technical explanation for functions and code blocks.

The Experiment-Weighted Attraction (**EWA**) model is an economic learning model introduced by Camerer & Ho (1999), which combines features of two other well-known economic learning models: Reinforcement Learning (**RL**), introduced by Erev & Roth (1998) as well as Belief Learning (**BL**) (including variations), based on the work of Cheung & Friedman (1997).

The introduced model works in any type of bi-matrix (strategic form) games. The current model is restricted to $2x2$ games but can be extended to include games with more players with code adjustments. The implementation of the code was based on the example Moffatt 2015, which deploys the EWA model simulation for the pursue/evade game, with a pursuer player and an evader player.

2. Notebook Structure

The core model is deployed on a Jupyter (Kluyver et al., 2016) file, which allows for an interactive combination of code, formulas, and descriptions. The model description and implementation are based in three parts: **Introduction to Economic Models of Learning**, **EWA Python Implementation** and **Simulation Data Exploration.** Each section is briefly described next.

2.1 Introduction to Economic Models of Learning

This section provides a theoretical description of the model's building blocks, including the two main components of the EWA formula: the update of attractions and calculation of strategy choice probabilities via logit transformation.

In addition, the program is supposed to read and interpret game matrices in a vector (linear) form, which is encoded in a python `list` data structure. The scheme for encoding a game is explained in this chapter and it is supposed to be followed when running the simulation for different games, as the payoff elements are indexed by positions.

2.2 EWA Python Implementation

The stepwise implementation of the EWA code is described in this section. The model starts by defining initial parameters and the target game for the simulation. The model will run multiple simulations for multiple rounds based on the following variables:

> `n_sims`: the number of simulations, that is, how many game-plays (interaction between two players) should be simulated
>
> `n_periods`: The number of rounds, which are iteration periods between each gameplay, that should be simulated

The remaining parameters and inputs necessary are described in detailed in the notebook file directly.

The program works by a master function that calls all the other functions to generate the simulation data. The functions are described as follows:

`probabilities_generation()` – applies the logit transformation in the attraction values, transforming them into normalized choice probabilities

`strategy_selection()` – perform the selection of a strategy among the available set of strategies from a player. This selection works by comparing the actual choice probabilities against a random uniform value, between 0 and 1.

`payoffs_generation()` – calculates the payoff values yielded for the selected strategies via matrix multiplication. The function calculates all possible payoffs given the players' choices scenario.

`weighted_payoffs_generation()` – generates a weighted version of the payoff values based on the EWA function, based on the `delta` parameter.

`ewa_attraction()` – executes the basic attraction update formula, for one player and one strategy type.

`ewa_update()` – executes the update of all attraction values for all possible strategies, applying the `ewa_attraction()` for all possible strategies.

`ewa_simulation()` – master function that executes the simulation model. This function controls all the other functions and yield as output the full simulated dataset for the selected parameter values.

The overall model workflow is based on the following sequential steps:

1. Initiate all necessary variables
2. generate choice probabilities (in round 1 they will be based on initial parameters; in the subsequent rounds they will be updated)
3. Select strategies for both players
4. Generate payoffs
5. Generate weighted payoffs
6. Sum acquired payoffs to previous rounds to get cumulative values
7. Calculate attractions
8. Parse datapoints in the specified structure

The simulated dataset is directly explored in the analysis section, described next.

2.2 Simulation Data Exploration

This section introduces a preliminary data analysis of the simulated model, which is focused on strategy frequencies, model parameters (attractions and experience) and payoffs. An important remark here is that the multi-plot grids applied in the analysis section are designed to show individual values for each simulation round. If too many simulation rounds are generated, this can add very high processing efforts to generate all the plots, so it is recommended to avoid the plot grids in case `n_sims` is high.

3. Data Output

The final output of the simulation model is a dataset containing all the variables generated during the process for all simulation rounds and pictures. Each of the variables is described in the notebook file, as in the table below:

Variable	Definition
sim_nr	Number of the simulation period. One simulation period can contain multiple game-playing rounds
round_nr	Number of simulated game-playing rounds
prob_p1s1	Probability of player one selecting the first strategy
prob_p1s2	Probability of player one selecting the second strategy
prob_p2s1	Probability of player two selecting the first strategy
prob_p2s2	Probability of player two selecting the second strategy
p1_strategy	Strategy selected by player one in the given round
p2_strategy	Strategy selected by player two in the given round
payoff_p1	Realized payoff for player one, based on the selected strategy
payoff_p2	Realized payoff for player one, based on the selected strategy
payoff_p1_s1	Payoff for player one's strategy one, given the choice of player two
payoff_p1_s2	Payoff for player one's strategy two, given the choice of player two
payoff_p2_s1	Payoff for player two's strategy one, given the choice of player one
payoff_p2_s2	Payoff for player two's strategy two, given the choice of player one
wp_p1s1	Weighted payoff for player one's first strategy
wp_p1s2	Weighted payoff for player one's second strategy
wp_p2s1	Weighted payoff for player two's first strategy
wp_p2s2	Weighted payoff for player two's second strategy
cum_payoff_p1	Cumulative payoff based on player one's decisions
cum_payoff_p2	Cumulative payoff based on player two's decisions
N	Experience level for the given round
A_p1s1	Attraction value for player one's strategy one in the given round
A_p1s2	Attraction value for player one's strategy two in the given round
A_p2s1	Attraction value for player two's strategy one in the given round
A_p2s2	Attraction value for player two's strategy two in the given round

A csv version of the data can be obtained directly by uncommenting the line `#simulation_df.to_csv("simulation_data.csv",  index=False)` in the dataframe object creation command.

References Used in the Model Construction

Baddeley, M. (2018). Behavioural economics and finance. Routledge. Chapter 5, p. 74-96.

Camerer, C., & Hua Ho, T. (1999). Experience-weighted attraction learning in normal form games. Econometrica, 67(4), 827-874

Cheung, Y. W., & Friedman, D. (1997). Individual learning in normal form games: Some laboratory results. Games and economic behavior, 19(1), 46-76.

Erev, I., & Roth, A. E. (1998). Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria. American economic review, 848-881.

Ho, T. H., Camerer, C. F., & Chong, J. K. (2007). Self-tuning experience weighted attraction learning in games. Journal of economic theory, 133(1), 177-198.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., and Willing, C. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows. In Loizides, F. and Schmidt, B., editors, Positioning and Power in Academic Publishing: Players, Agents and Agendas, pages 87 – 90. IOS Press.

Moffatt, P. G. (2015). Experimetrics: Econometrics for experimental economics. Macmillan International Higher Education. Chapter 18, p. 419-440.

Tanimoto, J. (2015). Fundamentals of evolutionary game theory and its applications. Springer Japan.