CS 240 : Lab 12 - Equilibria in Normal Form Games

TAs : Harshvardhan Agarwal, Pulkit Agarwal

Instructions

- This lab will be **graded**. The weightage of each question is provided in this PDF.
- Please read the problem statement and the submission guidelines carefully.
- All code fragments need to be written within the `TODO` blocks in the given Python files. Do not change any other part of the code unless stated otherwise.
- **Do not** add any **additional** *print* statements to the final submission since the submission will be evaluated automatically.
- For any doubts or questions, please contact either the TA assigned to your lab group or one of the 2 TAs involved in making the lab.
- The deadline for this lab is **Monday, 8 April, 5 PM**.
- The submissions will be checked for plagiarism, and any form of cheating will be penalized.

The submissions will be on Gradescope. You need to upload the following Python file: `q1.py`. In Gradescope, you can directly submit these Python files using the upload option (you can either drag and drop or upload by browsing). No need to create a tar or zip file.

1 Dominant and Nash Equilibria in NFGs [100 marks]

We work with 4 kinds of equilibria: Strongly Dominant Strategy Equilibrium, Weakly Dominant Strategy Equilibrium, Pure Strategy Nash Equilibrium, and Mixed Strategy Nash Equilibrium.

1.1 SDSE

- In a game with two players, Player 1 and Player 2, a **strictly dominant strategy equilibrium** occurs when each player's strategy is strictly dominant over all other strategies, regardless of the opponent's strategy.
- Mathematically, let S_1 and S_2 be the strategy sets of Player 1 and Player 2, respectively. If there exists some strategy $s_{1i} \in S_1$ such that the payoff for Player 1 from playing s_{1i} is strictly greater than the payoff from any other strategy in S_1 irrespective of the strategy played by Player 2, and similarly, there exists an analogous strategy s_{2j} for Player 2, then the outcome (s_{1i}, s_{2j}) is a strictly dominant strategy equilibrium.
- $u_1(s_{1i}, s_2) > u_1(s_{1k}, s_2), \forall s_2 \in S_2, s_{1k} \in S_1, i \neq k$
- $u_2(s_1, s_{2j}) > u_2(s_1, s_{2k}), \forall s_1 \in S_1, s_{2k} \in S_2, j \neq k$

1.2 WDSE

- A **weakly dominant strategy equilibrium** occurs when each player's strategy is at least as good as all other strategies, regardless of the opponent's strategy, but not strictly better in all cases (in which case it is an SDSE).
- Mathematically, similar to SDSE, but the strict inequality is replaced with a weak inequality.
- $u_1(s_{1i}, s_{2j}) \geq u_1(s_{1k}, s_{2j}), \forall s_2 \in S_2, s_{1k} \in S_1, i \neq k$
- $u_2(s_{1i}, s_{2j}) \geq u_2(s_{1i}, s_{2k}), \forall s_1 \in S_1, s_{2k} \in S_2, j \neq k$

1.3 PSNE

- A **pure strategy Nash equilibrium** occurs when each player's strategy is the best response to the strategies chosen by the other players.
- Mathematically, let s_1^* be the strategy of Player 1 and s_2^* be the strategy of Player 2. Then (s_1^*, s_2^*) is a pure strategy Nash equilibrium if and only if:
- $u_1(s_1^*, s_2^*) \geq u_1(s_1, s_2^*), \forall s_1 \in S_1$ and $u_2(s_1^*, s_2^*) \geq u_2(s_1^*, s_2), \forall s_2 \in S_2$

1.4 MSNE

- A mixed strategy Nash equilibrium occurs when players randomize their strategies, such that no player benefits from unilaterally changing their strategy.

- You are tasked with **finding an MSNE only when there are 2 players each with 2 strategies**. You may use any algorithm you find useful to do so (however, no external libraries are allowed). A nice way to solve the problem for the 2-player, 2-strategy case is described here.
- Mathematically, let $\sigma_1^* = (p_1^*, p_2^*)$ be the mixed strategy for Player 1, and similarly define σ_2^*. Let $\Pi[S]$ denote the probability distribution over a set S. Then (σ_1^*, σ_2^*) is a mixed strategy Nash equilibrium if and only if:
- $u_1(\sigma_1^*, \sigma_2^*) \geq u_1(\sigma_1, \sigma_2^*), \forall \sigma_1 \in \Pi[S_1]$ and $u_2(\sigma_1^*, \sigma_2^*) \geq u_2(\sigma_1^*, \sigma_2), \forall \sigma_2 \in \Pi[S_2]$

1.5 NFG Format

For test cases, we use the `nfg` file format. This file format defines a strategic N-player game, where the payoffs are listed in a tabular format. Please see this to know how the test cases are formatted.

1.6 Tasks

You have the following tasks to complete in `q1.py`:

1. In `find_strongly_dominant_eq (playerno, totalplayer, topplayer)` : [25 marks]

   For any player (`playerno`), return the index of the strategy that the player uses in the SDSE. The **convention of the index starts from 0**. If it's a 2-player game and there are 2 strategies for Player 1 and 3 strategies for Player 2 and the SDSE is strategy #1 for Player 1 and strategy #3 for Player 2, then return 0 in this function for Player 1 and return 2 in this function for player 2.

   Partial code is provided for the case when there are remaining players left to consider. You only need to write the code for when `len(totalplayer)=0`.

2. In `find_weakly_dominant_eq (playerno, totalplayer, topplayer)` : [25 marks]

   For each player (`playerno`), return the index (`eqindex`) of the strategy that the player uses in the WDSE. The other details are the same as that for SDSE above.

3. In `psne_gen (num_players, strategy, util_matrix)` : [30 marks]

   Return the list of PSNEs for this game as a 2D array. For example, for the following game,

   ```
   NFG 1 R "NFG game"
   { "Player 1" "Player 2" "Player 3" } { 2 2 2 }

   2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
   ```

   The output list should be

   $$[[0, 0, 0], [1, 1, 1]]$$

   with the PSNEs being

   (a) All 3 players playing the 1st strategy

(b) All 3 players playing the 2nd strategy.

4. In `msne_gen (num_players, strategy, util_matrix)` : [20 marks]

   Return any MSNE for this game as a 2D array. For example, for the following game,

   ```
   NFG 1 R "NFG game"
   { "Player 1" "Player 2" } { 2 2 }

   3 2 0 0 0 0 2 3
   ```

   The output list should be

   $$[[0.6, 0.4], [0.4, 0.6]]$$

   with the MSNE being

   (a) Player 1 plays the 1st strategy with probability $3/5$.

   (b) Player 2 plays the 1st strategy with probability $2/5$.

   Note that if there is an MSNE in which no probability is zero, then you must return that MSNE. Else you can return the MSNE with some zero probabilities as well. It is guaranteed that there will be at least one MSNE for the given game.

   IMPORTANT: You only need to solve this part for the case when the number of players is 2, and each player has 2 strategies. You can use these values directly as well.

The Python file needs to be called as `python q1.py input/ex.nfg`, with the input being different `nfg` files from the `input` folder.

1.7 Code

1.7.1 Main Function

- The `nfg` file is read by the code and converted into 2 lists - `strategies` and `gamedata`. `strategies` contains the number of strategies of each player (the info from the 2nd line of the `nfg` datafile). `gamedata` is the list of payoffs (the info from the 3rd line).
- Strongly dominant strategies are checked and the function is called for each player.
- If no SDSE exists, weakly dominant strategies are checked and the respective function is called for each player.
- At the end, the PSNE and MSNE functions are called. Note that the MSNE has to be found only when there are 2 players, and each of them has 2 strategies.

1.7.2 Helper Functions

- `make_util_matrix (num_players, strategy, util_list)`
  - Constructs a utility matrix based on the number of players, their strategies, and a list of utilities.
  - This matrix is used when finding the PSNEs and MSNE.
  - You are free to use this helper function anywhere as needed.
- `make_allpermut (num_players, strategy)`
  - Provides a list of all possible permutations of strategies
- `check (array, target, forstrat, num_players, strategy, util_matrix)`
  - Checks if a given strategy is the best response to a target strategy for all players
  - Returns 1 if the given strategy is the best response, otherwise 0.
- `sel_index (player, args, multiplier, num_players)`
  - Selects the index in the game data based on the player and their chosen strategies.
- `intersect (a, b)`
  - Finds the intersection of two lists.

The details of arguments and return values are provided in comment blocks below each function.