ARISTOTLE
UNIVERSITY
OF THESSALONIKI

Department of Electrical
and Computer Engineering

Group Project for the Game Theory Course

Evolutionary Games Toolbox

Group Number	2
Delopoulos Emmanouil	entelopo@ece.auth.gr
Vaporis Dimitrios	dvaporis@ece.auth.gr
Voulkidis Vlasios	vlasiosv@ece.auth.gr
Professor	Kehagias Athanasios

May 2025

Contents

1 Introduction

1.1 Description of the problem

This is the report for the project of the course “Game Theory” of the Department of Electrical and Computer Engineering of the Aristotle University of Thessaloniki. The subject of this project is the implementation and study of the Axelrod evolutionary tournament in the game ”Prisoner’s Dilemma.” This term refers to a social dilemma, that is, any game of the form $\begin{bmatrix} R & S \\ T & P \end{bmatrix}$, where the following condition holds: $S < P < R < T$. Each player is given a choice, whether to cooperate or to defect. The payoff is calculated based on the player’s move as well as the opponent’s move; if both players cooperate, they each get R, if one cooperates and the other defects, the defector receives T and the cooperator receives S, otherwise if they both defect they both get rewarded P points. Paradoxically, a rational player may notice that no matter what the opponent plays, it is always better to defect, thus creating a Nash equilibrium (a game outcome that no player has motive to move away from) for two perfectly rational players in the “Defect-Defect” zone, making them both miss out on possible extra points, were they to cooperate.

The repeated iteration of this game creates a match between strategies; multiple matches between strategies form a tournament, and the computation of a new population based on each strategy’s performance in the tournament constitutes the main focus of the study, an evolutionary tournament.

The project consists of four main functions, each of which performs a specific task that can be characterized by two features of the respective assumption: the nature of the simulation (theoretical or real) and the evolutionary dynamic (fitness or imitation). Each function is introduced individually, with an explanation of its operation and any assumptions made in each case. The first part of the project is largely based on the study by Mathieu et al. In fact, an effort was made to replicate the results as closely as possible to those presented in their 1999 paper.

For the study of the project, some concepts are crucial to understand. More specifically:

1. A match is the base game played between two players, with a specified number of rounds (amount of times the players choose a move) which is unknown to the players.
2. A strategy is a specific algorithm followed by a player in order to calculate their next move in a match.
3. A population is the number of players following each strategy in a given tournament.

4. A tournament consists of all possible matches between each pair of players (players do not play against themselves though).

5. An evolutionary game is the repetition of a tournament for a specified number of generations, where the population of each generation is calculated based on specified evolutionary dynamics.

As mentioned above, the evolution dynamics studied in this project are the following:

1. Fitness dynamics, where for each generation the number of players adopting each strategy is proportional to the performance (measured by a specific metric, in this case the total payoff in the tournament) of that strategy.

2. Imitation dynamics, where after each generation a specified number of players adopt the best strategy (calculated either as an Individual or as a Total, more on that on a later chapter) of the previous generation.

1.2 Related Work

The subject of the project was in large popularized by the famous Axelrod Tournament[3][2], where many famous game theorists were challenged to submit strategies for the Iterated Prisoner's Dilemma. The winning strategy ended up being the well-known Tit For Tat strategy, which, despite its simpleness, had many redeeming qualities; it was nice, meaning it was never the first to defect. It was retaliatory, meaning it was ready to counter attack if the opponent defects. It was also forgiving, meaning it did not hold a grudge too long; if an opponent defected but repented, Tit For Tat forgave them and continued to cooperate. Lastly, it was clear; it is not difficult for the opponent to understand the intentions of the strategy, thus making it more likely that cooperation occurs.

Since 1980, when the tournament was held, a large number of studies have been published on the matter, a lot of which have focused on the evolutionary aspect of the tournament, trying to mimic the actual evolution of species in the animal kingdom, as well as the birth of cooperation between humans. One such paper was published in 1999 by Philippe Mathieu, Bruno Beaufils and Jean-Paul Delahaye with title "Studies on Dynamics in the Classical Iterated Prisoner's Dilemma with Few Strategies"[4]. In this specific paper, the evolutionary dynamics analyzed were Fitness Dynamics and the results aimed to showcase the different possible forms of dynamics that can occur and which aspects of the simulation they may depend on. As mentioned previously, the first part of this project is based exactly on that paper, aiming to recreate the results presented by Mathieu et al, as well as slightly modifying the dynamics in order to seek possible differences in the resulting dynamics.

2 Quick Start

To begin using the toolbox:

1. **Clone the repository:**

```
git clone https://github.com/mdelopo/EvolutionaryGamesToolbox.git
cd EvolutionaryGamesToolbox
```

2. **Run the setup script in MATLAB:**

```
setup
```

3. **Run example scripts:**

All example scripts contained in the Examples folder can be run immediately. Each script includes comments and setup instructions to help you understand and adapt them to your needs.

For the full source code, documentation, and examples, visit the GitHub repository:
https://github.com/mdelopo/EvolutionaryGamesToolbox

(A detailed description of the code structure and function is provided in the Appendix.)

3 Fitness Dynamics

The first evolutionary dynamics to be analyzed (and the one studied by Mathieu et al.) are the **Fitness Dynamics**, according to which each strategy in the simulation is assigned a score, which is then used to calculate the population distribution for the next generation. The approach followed in both functions that use fitness is that of Mathieu et al., where the fitness of each strategy is calculated as follows:

Suppose the population consists of 3 strategies, A, B, and C. Based on the game matrix B, the number of rounds per match T, and, of course, the strategies that face each other in each case, the payoffs for each of the two strategies are calculated and stored as, for example, $V(A|B)$, the payoff of strategy A when it faces strategy B. Then, the score for each player of generation n using a given strategy (and thus, essentially, the score of the strategy itself) is calculated as, for example, for strategy A:

$$g_n(A) = W_n(A)V(A|A) + W_n(B)V(A|B) + W_n(C)V(A|C) - V(A|A)$$

where $W_n(A)$, $W_n(B)$, $W_n(C)$ are the population sizes of each strategy in generation n. (Note that the payoff for playing against one's own strategy is subtracted once, because it is assumed that individuals do not play against themselves.)

Finally, the total tournament score is calculated as:

$$t(n) = W_n(A)g_n(A) + W_n(B)g_n(B) + W_n(C)g_n(C)$$

and the population in generation $n + 1$ for strategy A becomes:

$$W_{n+1}(A) = \frac{\Pi W_n(A)g_n(A)}{t(n)}$$

It should be emphasized that this logic is applied due to the fully deterministic nature of the implemented strategies. If there were strategies with random elements, the theoretical calculation of the outcome of each match would be impossible, and one would instead need to compute some expected value—something that falls outside the scope of this study.

3.1 The function TourSimFit

The first function implemented is [POP, BST, FIT] = TourSimFit(B, Strategies, POP0, T, J, compensation), where B is the payoff matrix of the game, Strategies is an array of strings with the names of the strategies participating in the simulation, POP0 is the initial population, T is the number of rounds in each match, and J is the number of generations of the evolutionary tournament. Additionally, compensation is an optional boolean argument (the function can be called without including it), whose function is described below. The function returns the following:

- POP: a matrix with the population of each strategy per generation,
- BST: a matrix indicating the best strategies in each round (0 if not among the best, 1 if among the best — ties are counted),
- FIT: a matrix with the fitness scores of each strategy for each generation.

The logic of the function follows what was presented earlier, but an additional mechanism is added during the computation of the next generation's populations to avoid decimal values and ensure that the population for each strategy in each generation remains an integer. The logic works as follows:

The initial result of the population computations is taken as the floor of the decimal values. Then, any deficit that arises from applying the floor function is computed, and one new player at a time is randomly assigned to some strategy, until the deficit is eliminated. A check is also performed to ensure that the strategy does not already have a population of zero, to avoid its "revival." In this way, the total player population remains constant throughout the simulation, as is assumed by Mathieu et al.

This choice is deliberate. Cases were tested in which each player was assigned to the strategy whose calculated population was closest to the next integer (e.g., if a strategy had an initial computed population of 199.8, it would be rounded to 200), as well as the opposite case, where players were assigned to the strategy furthest from the next integer. These algorithms showed some undesirable behaviors, such as maintaining a very small number of players in certain strategies (e.g., a strategy remaining stuck at population 2 because the next generation's computed population is 1.8, or 1.1 respectively). By preserving this random element, the simulation results do not differ significantly from those without randomness, while the method guarantees that weaker strategies are completely eliminated over time.

However, observing the deviation of this implementation from the results in the paper, we added the following adjustment: an extra boolean argument in the function TourSimFit, named compensation with default value false, which performs the functionality described below. With value false, it returns results using only simple rounding to the nearest lower integer (floor rounding), which we believe is also the method used in the paper. In this case, the population per generation is not necessarily constant, but the deviation does not increase additively — perhaps just 1 or 2 individuals are lost in some generations due to rounding down. With value true, it returns results according to the player-reassignment logic described earlier. Below the main loop for the tournament simulation, including the possible compensation of the deficiency in the population, is presented with pseudocode.

Algorithm 1 TourSimFit Simulation

for $i = 1$ to J **do**
  for $j = 1$ to N_{strat} **do**
    $FIT[i][j] \leftarrow 0$
    for $k = 1$ to N_{strat} **do**
      $FIT[i][j] \leftarrow FIT[i][j] + \text{payoff}[j][k] \cdot POP[i][k]$
    end for
    $FIT[i][j] \leftarrow FIT[i][j] - \text{payoff}[j][j]$
  end for
  $maxVal \leftarrow \max(FIT[i][:])$
  $allMaxIndices \leftarrow \{j \mid FIT[i][j] = maxVal\}$
  for all $j \in allMaxIndices$ **do**
    $BST[i][j] \leftarrow 1$
  end for
  $total\_each \leftarrow POP[i][:] \cdot FIT[i][:]$ {Element-wise multiplication}
  $total \leftarrow \sum total\_each$
  for $j = 1$ to N_{strat} **do**
    $POP[i+1][j] \leftarrow \left\lfloor POP[i][j] \cdot \dfrac{FIT[i][j]}{total} \cdot N \right\rfloor$
  end for
  if *compensation* is **true then**
    $N_{new} \leftarrow \sum POP[i+1][:]$
    $deficiency \leftarrow N - N_{new}$
    while $deficiency > 0$ **do**
      $k \leftarrow$ random integer in $[1, N_{\text{strat}}]$
      if $POP[i][k] = 0$ **then**
        continue
      end if
      $POP[i+1][k] \leftarrow POP[i+1][k] + 1$
      $deficiency \leftarrow deficiency - 1$
    end while
  end if
end for

In the simulations that follow, a comparison is also presented between these two methods and the results of the function TourTheFit, which is described below. Including the results from the paper was deemed unnecessary.

3.2 The function TourTheFit

The second function implemented is [POP,BST,FIT] = TourTheFit (B, Strategies,-POP0, T, J), with arguments and outputs entirely identical to those of the previous one. The only substantial difference lies in the fact that now, due to the theoretical nature of the simulation, the part of the code responsible for maintaining integer values in the population is omitted, since decimal numbers are allowed, as the population does not actually consist of individual players. Below, the main loop of the evolutionary tournament is presented via pseudocode.

Algorithm 2 TourTheFit Simulation

for $i = 1$ to J **do**
  for $j = 1$ to N_{strat} **do**
    $FIT[i][j] \leftarrow 0$
    for $k = 1$ to N_{strat} **do**
      $FIT[i][j] \leftarrow FIT[i][j] + \text{payoff}[j][k] \cdot POP[i][k]$
    end for
    $FIT[i][j] \leftarrow FIT[i][j] - \text{payoff}[j][j]$
  end for
  $maxVal \leftarrow \max(FIT[i][:])$
  $allMaxIndices \leftarrow \{j \mid FIT[i][j] = maxVal\}$
  for all $j \in allMaxIndices$ **do**
    $BST[i][j] \leftarrow 1$
  end for
  $total\_each \leftarrow POP[i][:] \cdot FIT[i][:]$ {Element-wise multiplication}
  $total \leftarrow \sum total\_each$
  for $j = 1$ to N_{strat} **do**
    $POP[i+1][j] \leftarrow POP[i][j] \cdot \dfrac{FIT[i][j]}{total} \cdot N$
  end for
end for

3.3 Simulations - Examples

From this point on, the simulations of the fitness dynamics are presented, using the functions discussed above. Each plot presented is created by the corresponding example referenced in the text, contained in the Examples folder of the GitHub repository. For a more detailed description of each call and the parameters used in each example, please refer to the Documentation.pdf file contained in the Documentation folder of the repository, as well as the actual source code of each example. Along with the plots created by the functions described, the plots of the original 1999 paper by Mathieu et al are presented, so as to allow direct comparison and discussion of every example.

3.3.1 1st Simulation - Defectors may be strong

Already from the first simulation (Figure 1), an interesting result emerges. The purpose of this simulation is to demonstrate that, in specific sets of strategies and favorable initial population values, defecting strategies such as per_ddc are capable of dominating. However, the theoretical analysis of the scenario studied in the paper does not present exactly the same picture. This is due to the fact that the population of soft_majo in the simulation cases completely disappears around the 20th generation, whereas in the theoretical case, this does not occur. Thus, after a few generations and a relative decrease in the population of per_ddc, the strategy soft_majo returns and the strategy Alternator develops, which is essentially the perfect opponent of soft_majo, as it defects exactly as much as necessary to keep soft_majo cooperating. After a certain point, of course, both of these populations begin to decline again, while the population of per_ddc increases, which leads us to suspect that this example theoretically results in oscillation. In the simulations, however—in both the compensation case and the one without it—the results are very similar to those in the paper: the slight variation in the compensation case is due to the stochastic element, which makes the curve appear less smooth than in the case without compensation. Run example01 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.2 2nd Simulation - Monotonous Convergence

From now on, the results are presented for the simulations used by Mathieu et al. to classify the graphs into 5 categories, the first of which (Figure 2) is the monotonous convergence of populations. This is the most common form that emerges, as the oscillations presented below are quite sensitive to initial conditions. The results of both the theoretical analysis and the two simulations are identical to those of Mathieu et al., with the only difference being a slightly different final population in the case of compensation, which is again due to the stochastic element. However, the ranking of the strategies remains the same. Run example02 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.3 3rd Simulation - Attenuated Oscillatory movements

The next category of results from the paper is the diminishing oscillations of populations (Figure 3). The results in this case are identical to those of Mathieu et al. for all three cases presented. The initial populations in this particular case are quite large, so the one or two players that randomly oppose in the compensation scenario do not significantly affect the trajectory of the function. Run example03 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.4 4th Simulation - Periodic movements

The third case is the appearance of periodic movements/oscillations (Figure 4), without an increase or decrease in the amplitude of the oscillations. This behavior generally appears when there are three strategies with a ”rock-paper-scissors” logic: the 1st ”beats” the 2nd, which ”beats” the 3rd, which in turn ”beats” the 1st. In this particular case, per_ccd beats soft_majo, which beats per_ddc, which beats per_ccd. With appropriate initial populations, the following results are observed. The theoretical analysis shows that the oscillation actually fades — it is a diminishing oscillation as before. This is due to the fact that in discrete cases, such as these simulations, it becomes certain through suitable choices that the system will return to a previous state, and thus repetition/oscillation will occur. However, in the theoretical analysis, this does not happen (due to the ”infinity” of states because of decimal numerics), and thus the oscillation is damped. The TourSimFit function without compensation again yields results identical to those of Mathieu et al., while the case with compensation, although visually less appealing, also captures the oscillation, even with noticeable noise due to randomness (the amplitudes of the oscillations are small, so even the addition of one or two extra players has a noticeable effect). Run example04 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.5 5th Simulation - Increasing oscillations

One of the most unexpected cases is that of increasing oscillations (Figure 5). With an appropriate choice of the payoff matrix, strategies, and initial populations, this phenomenon can be observed.

A result very similar to that of the paper was observed in the case of TourSimFit without compensation. However, for the cases of TourTheFit and TourSimFit with compensation, diminishing oscillations are observed, revealing two truths about the cases of increasing oscillations. First, they arise from populations capable of exhibiting normal oscillations, with a suitable modification of the payoff matrix. Second, they are particularly sensitive cases that collapse back into normal or diminishing oscillations with even a slight modification of the dynamics (such as the logic of compensation). It is quite possible that for an appropriate choice of B, the cases of TourTheFit and TourSimFit with compensation could also transform into increasing oscillations, but it was considered more important for this work to present this divergence in the results. Run example05 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.6 6th Simulation - Chaos/Disordered oscillations

The last case presented in the paper is that of disordered oscillations (Figure 6). The authors of the paper rightly hesitate to characterize it as a truly chaotic case because, due to the discrete nature of the simulation, any such behavior after a sufficient number of generations either reaches equilibrium or simply repeats. However, the results of this particular simulation appear quite chaotic. Again, the case of TourSimFit without compensation fully matches the results of the paper. In contrast, due to the sensitivity of the phenomenon, the cases of the theoretical analysis and the actual simulation with compensation differ significantly, as they do not exhibit the chaotic behavior around generation 140 as in the case of the paper. Nevertheless, they predict the survival of the per_ccccd and Prober strategies, as well as the values toward which the populations in the TourSimFit case without simulation seem to tend. Run example06 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.7 7th Simulation - Sensitivity of dynamics to population's size - First Simulation

From now on, the simulations showcase the sensitivity of the results to the initial conditions of the examples. Again, the results presented are compared to the results of Mathieu et al and any differences are discussed. The first simulation aims to present the sensitivity of the dynamics to the initial population's size for each strategy. More specifically, the same set of strategies can lead to many different forms of dynamics occuring, depending on the population of each of the strategy. This should not come as a surprise, as even in the previous examples the strategies used were often the same, with the differences in dynamics being caused by different initial populations. In this first example (Figure 7), the result is a periodic movement, which is also recreated in the TourSimFit without compensation, as well as (mostly) in the TourSimFit with compensation. However, the theoretical result of TourTheFit is an attenuated oscillation, as in the case of the Figure 4, for the same reasons as before. Run example07 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.8 8th Simulation - Sensitivity of dynamics to population's size - Second Simulation

The second example of the showcase of the sensitivity of the dynamics to the population's size shows a monotonous convergence that is caused by a slight differentiation in the initial population vector of the previous simulation. The only difference between this simulation and the previous one is one extra per_ddc added, which ends up being enough to change the resulting dynamics completely. This is recreated (Figure 8) in the case of TourSimFit without compensation. However, in the cases of TourTheFit there appears to still be a slight attenuated oscillation, meaning the convergence is not monotonous, and in the case of the TourSimFit with compensation the result is still a periodic movement, obviously caused by the random element of this method. Thus, we observe a large sensitivity of the dynamics to the repartition method of the population, which is also discussed in a later example. Run example08 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.9 9th Simulation - Sensitivity of winner to population's size - First Simulation

The second case presented is the sensitivity of the winner to the population's size, with winner being the strategy that ends up having the most members of the population at the end of the simulation. This may not be obvious at first; we have seen a lot of cases where the change in dynamics is clearly visible but the ensuing winner still remains the same in most cases. The first example (Figure 9) shows an initial population that ultimately announces per_ddc as the winning strategy; the winner is recreated by each of our own methods, however the dynamics in the case of TourTheFit are visibly different, due to the population of soft_majo never actually dying out and thus making the Alternator strategy more viable (as we covered, Alternator is the perfect counter to soft_majo). Run example09 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.10 10th Simulation - Sensitivity of winner to population's size - Second Simulation

The second part of the sensitivity of winner to population's size showcase adds one soft_majo participant to the initial population, making Alternator the winner. The results (Figure 10) are recreated in the case of TourSimFit without compensation. However, in the cases of TourTheFit and TourSimFit with compensation, Alternator does not seem to be the decisive winner of the simulation, since at the end of the simulation there is still a rising population of per_ddc, strategy which does better versus Alternator, meaning that the eventual winner would be per_ddc. This in turn is caused by the fact that the population of per_ddc is not eliminated in these two cases, thus making a comeback possible. Note that, as Mathieu et al suggest, the modified strategy never wins (unless if it was already the winner). Run example10 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.11 11th Simulation - Sensitivity to game length - First Simulation

The next aspect of initial conditions studied is the game length, meaning the number of rounds each match between two players lasts. As it turns out, periodic strategies like most studied in this project are sensitive to the game length, because of the varying “winner” of each match based on the amount of rounds played. In the first part, each match lasts 7 rounds and the result is a periodic movement. This is also recreated (Figure 11) in the case of TourSimFit without compensation, but it again becomes an attenuated oscillatory movement for the cases of TourTheFit and TourSimFit with compensation, for the same reasons as discussed multiple times (a true periodic movement without making rounding errors in some generations is very sensitive). Run example11 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.12 12th Simulation - Sensitivity to game length - Second Simulation

The second simulation of this part changes only the number of rounds played each match to 6. This results in an attenuated oscillatory movement, which is recreated (Figure 12) by every function of our own. Thus, it is shown that the dynamics, as well as the eventual winner (since a periodic movement like the one showcased in the previous example has no official winner) are sensitive to the game length, especially when periodic strategies are used. This happens because, for example, in a 6 round game $\mathrm{per\_ccd}$ does not lose as hard to $\mathrm{per\_ddc}$ as in the case of a 7 round game (it suffers one less “effective” defection, meaning an opponent defection met with cooperation). These slight modifications to the score of each match change the fitness score of each strategy enough to create visible changes to the outcome. Run example12 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.13 13th Simulation - Sensitivity to CIPD payoff - First Simulation

One other initial condition of the simulation that affects the results greatly is the CIPD payoff, meaning the payoff matrix B. This should not come as a surprise; the fitness of each strategy is calculated based on the scores of each match, which in turn is calculated by the payoff matrix. We have also seen in a previous example (Figure 5) how even a slight change to a single element of the payoff matrix can lead to different ensuing dynamics. In the first simulation, the payoff matrix

$$B = \begin{bmatrix} 3 & 0 \\ 4.6 & 1 \end{bmatrix}$$

is chosen, resulting in increasing oscillations. This is recreated (Figure 13) by all three of the functions, with a slight difference in the cases of TourTheFit and TourSimFit with compensation being that the oscillation ends quicker, as all the members of the population end up using the per_ccd strategy. Run example13 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.14 14th Simulation - Sensitivity to CIPD payoff - Second Simulation

In the second part of the comparison the payoff matrix is changed slightly to

$$B = \begin{bmatrix} 3 & 0 \\ 4.7 & 1 \end{bmatrix}$$

resulting in periodic movements. This behavior is also presented by all functions (Figure 14), thus showing that the payoff matrix is capable of changing the resulting dynamics even with a minor adjustment. Run example14 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.15 15th Simulation - Sensitivity to repartition computation method - First Simulation

The last aspect of the simulation that is capable of producing vastly different results if altered is the repartition computation method of the simulation, meaning the way the population of the next generation is calculated. This should already be clear, since the functions of the project already have showcased different results in multiple simulations. To start, Mathieu et al calculate the population of the next generation both by rounding to the lower integer and by keeping the float number calculated (thus creating the TourTheFit function). The results are presented in Figure 15; the results of the plots by Mathieu et al match the results of TourSimFit with compensation and TourTheFit accordingly. Note that TourSimFit with compensation creates a similar periodic movement to the case without compensation. Also note that both plots presented by Mathieu et al are given in the same figure this time. This happens because the difference being showcased is essentially the difference between TourTheFit and TourSimFit, which we present neatly with our Figure. Run example15 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.16 16th Simulation - Sensitivity to repartition computation method - Second Simulation

This final pair of simulations again showcases the sensitivity to the repartition computation method, this time by keeping the ratios between strategies constant but by changing the actual population each strategy initially has. The results (Figure 16) are exactly the same in all cases, also matching the results by Mathieu et al. Run example16 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.3.17 17th Simulation - Sensitivity to repartition computation method - Third Simulation

Lastly, all initial populations of the previous simulation are divided by 10 (thus keeping the ratios of strategies constant). This results in an increasing oscillation in the paper by Mathieu et al. The figure created (Figure 17) shows the same result in the case of TourSimFit wihout compensation, but in the other two cases the dynamics constructed remain attenuated oscillations. Thus, it is once again proven that the method used to calculate the population of the next generation is one of (if not the) most importants aspect in terms of how much the results may differ. Also, it is shown that the ratio between strategies is not always enough to determine the ensuing dynamics; in some cases, such as this, the dynamics are vastly different, just by changing the magnitudes of the populations, not their ratio (from attenuated to increasing oscillations). Run example17 of the Examples folder (after reading Quickstart guide) to recreate the figure.

3.4 Discussion

From the above, the following final conclusions emerge regarding the evolutionary Fitness Dynamics:

1. Mathieu et al. appear to have used a simple floor function when calculating new populations after each generation, as this method yielded results closest to ours.

2. In many cases of generally unstable behaviors, the diagrams differ to an observable degree depending on whether the function TourTheFit or the function TourSimFit is used, with or without compensation. This is due to the fact that many of the resulting diagrams are highly sensitive, and even a small change in the dynamics logic can significantly alter the results.

3. The resulting diagram depends on the initial population values, the strategies used, and the payoff matrix. Even if two of these factors remain the same, a change in one of them can lead to drastically different results (even different rankings, as seen in Figure 5).

4. Assigning the remaining population members randomly according to the logic of compensation seems to be a good choice, especially given its simplicity, as it does not significantly alter the results (strategies that theoretically die out actually do die out; in general, the population tendencies do

not change). Depending on the specific case being examined, it resembles either the TourTheFit case or the TourSimFit case without compensation.

5. In general, the discrete nature of the TourSimFit simulations, with or without compensation, leads over time either to recurring oscillations or convergence to certain final population values. Chaos cannot really be achieved; even in Figure 6, the populations eventually converge to the values predicted by TourTheFit. However, this does not prevent us from generating the interesting results seen in the paper.

6. The results depend on many factors including the game length, the payoff matrix, the initial population (both ratio and magnitude) and the repartition computation method.

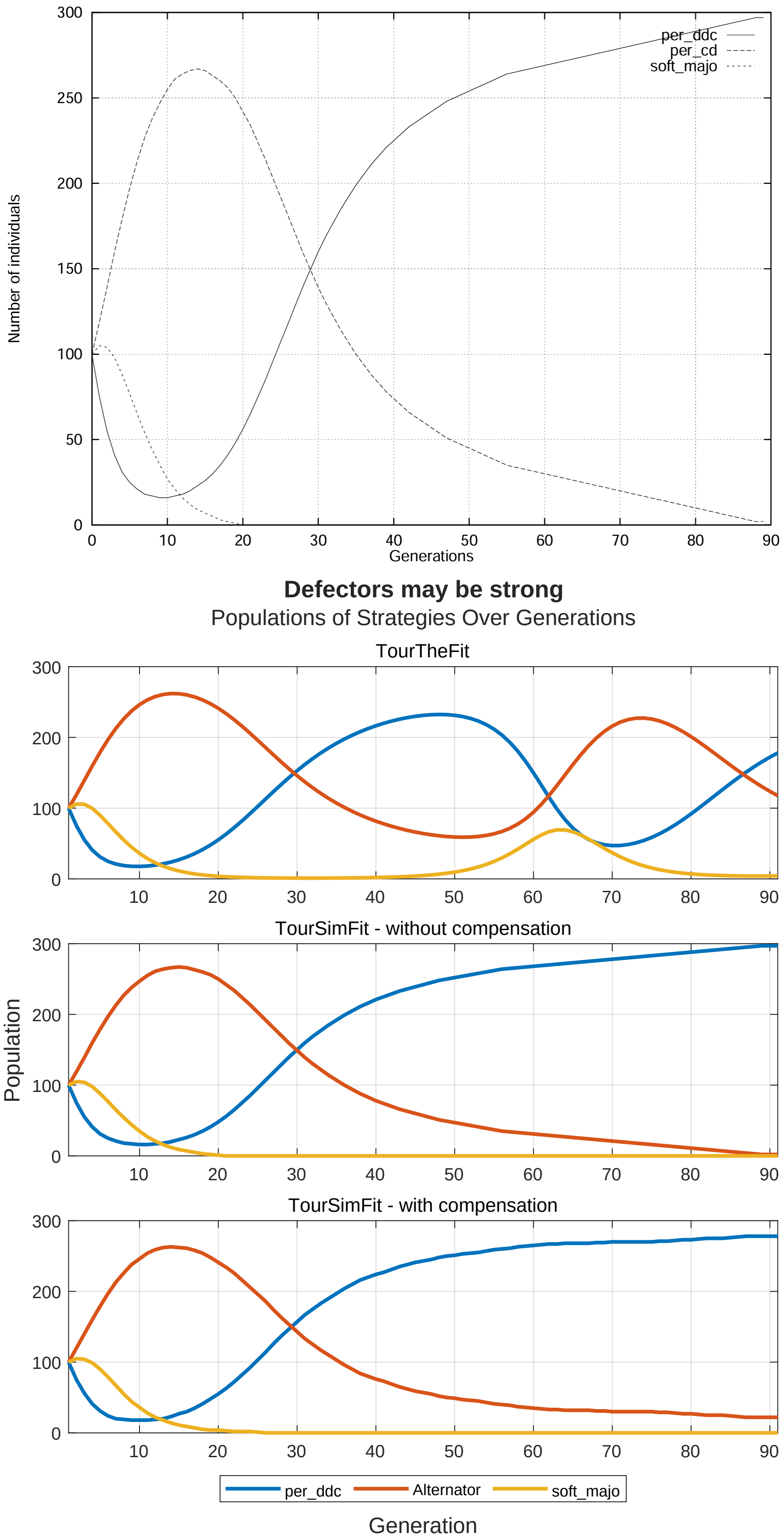


Figure 1: 1st Simulation - Defectors may be strong

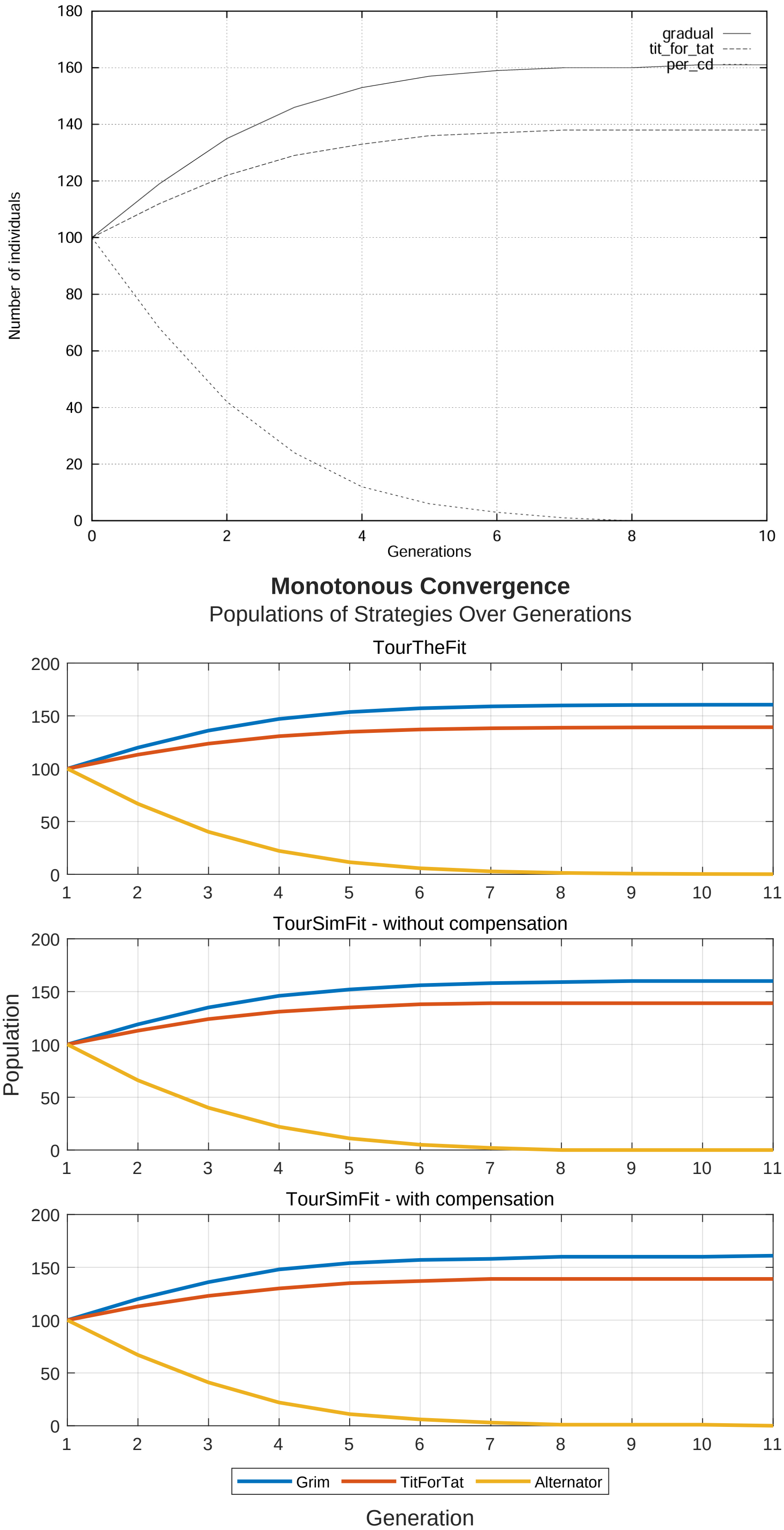
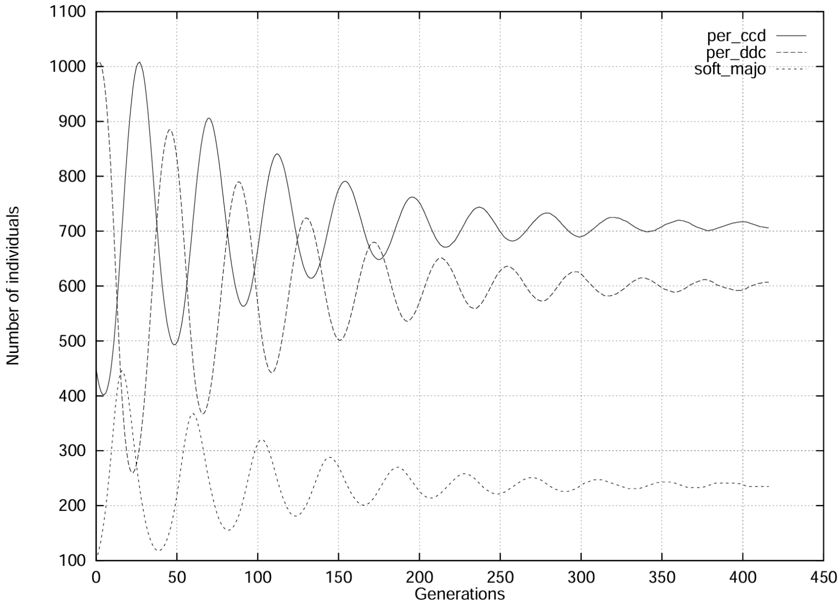


Figure 2: 2nd Simulation - Monotonous Convergence



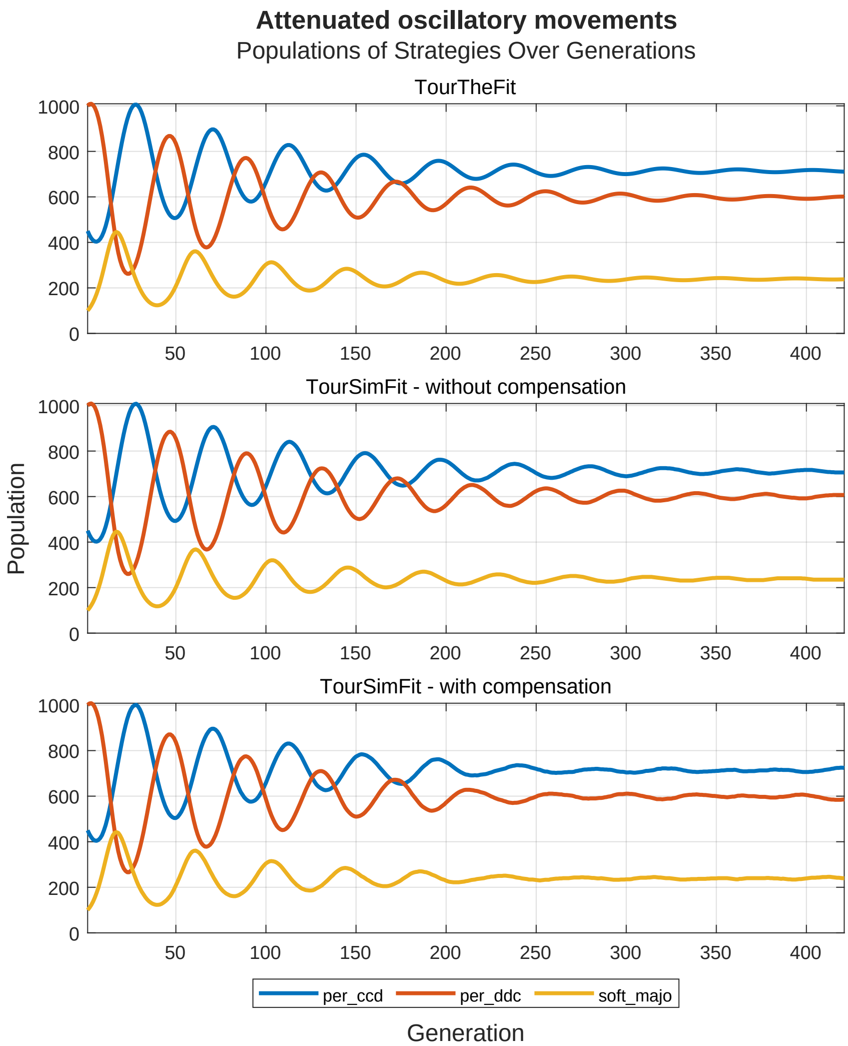


Figure 3: 3rd Simulation - Attenuated oscillatory movements

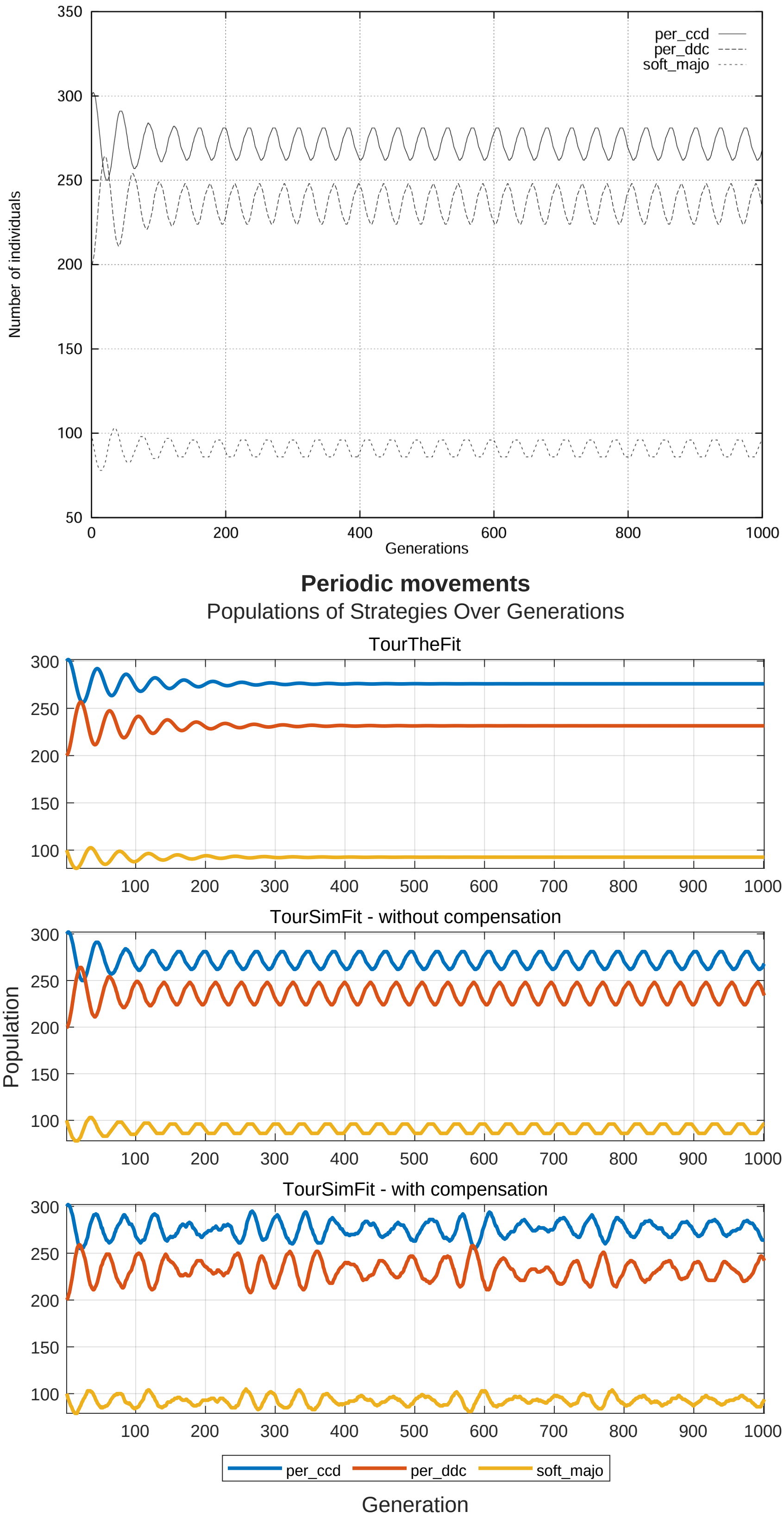
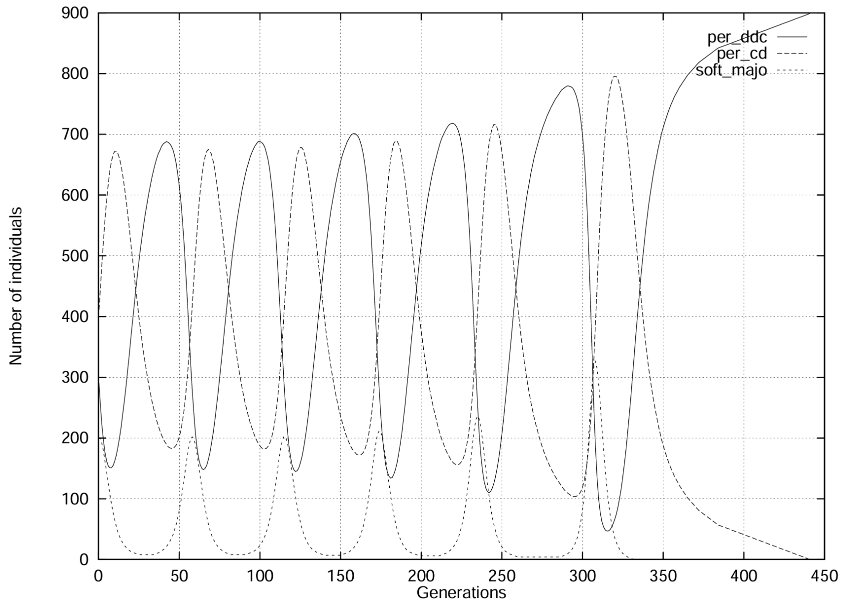


Figure 4: 4th Simulation - Periodic movements



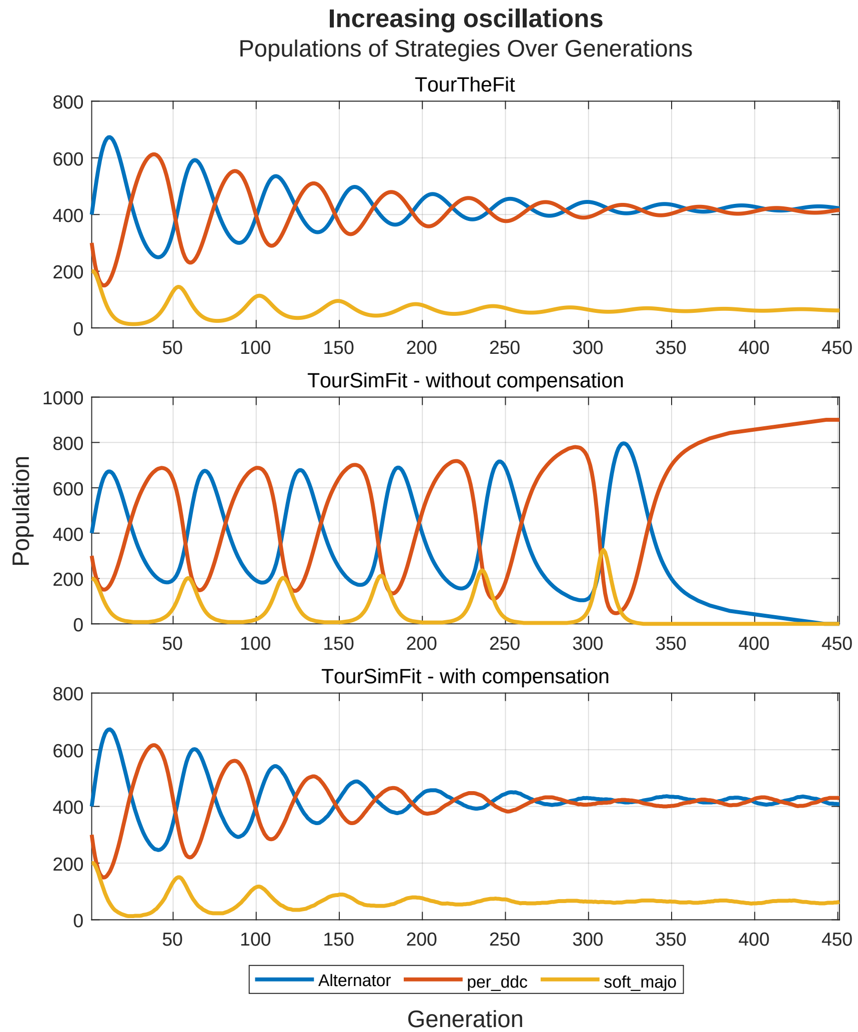


Figure 5: 5th Simulation - Increasing oscillations

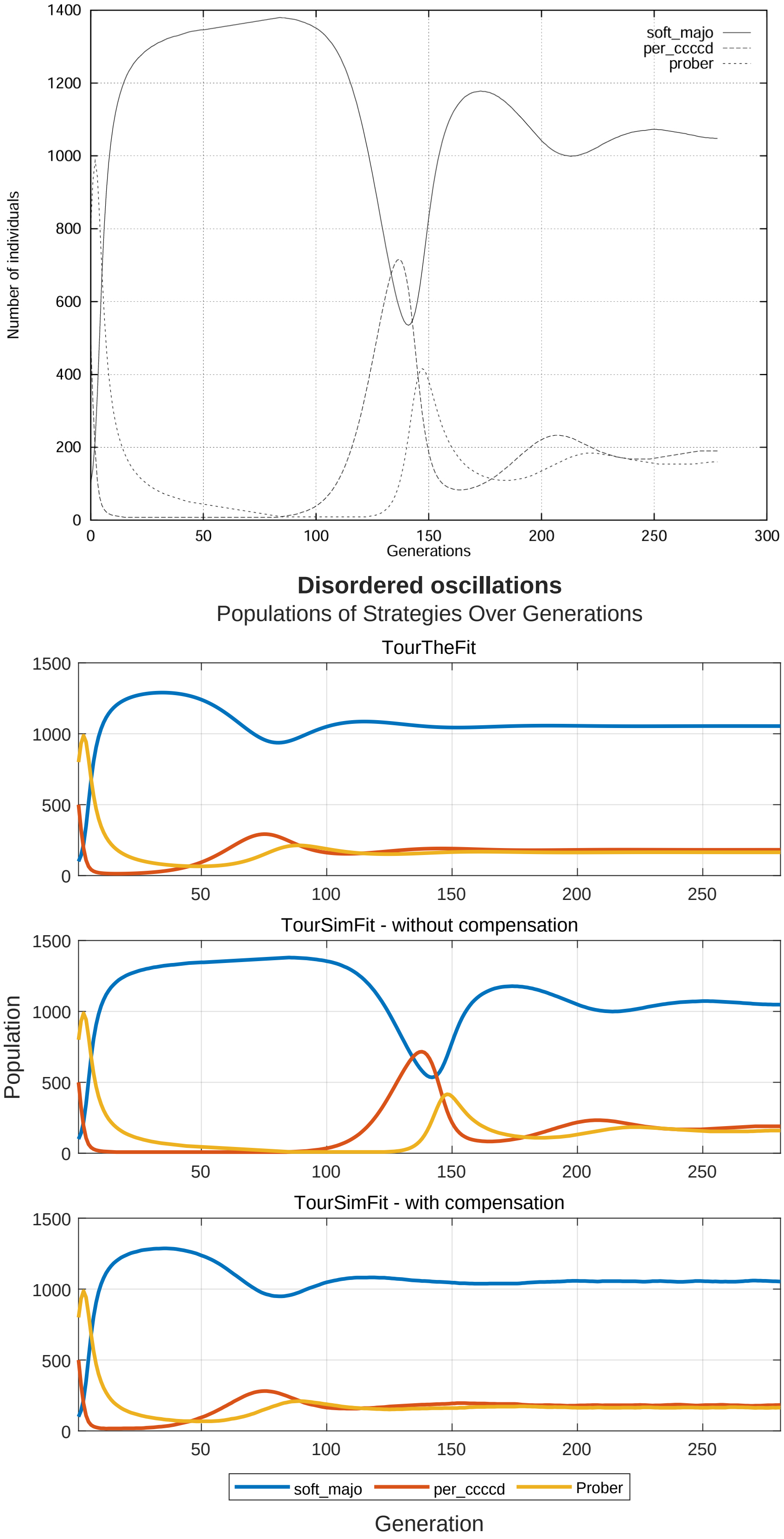


Figure 6: 6th Simulation - Disordered oscillations

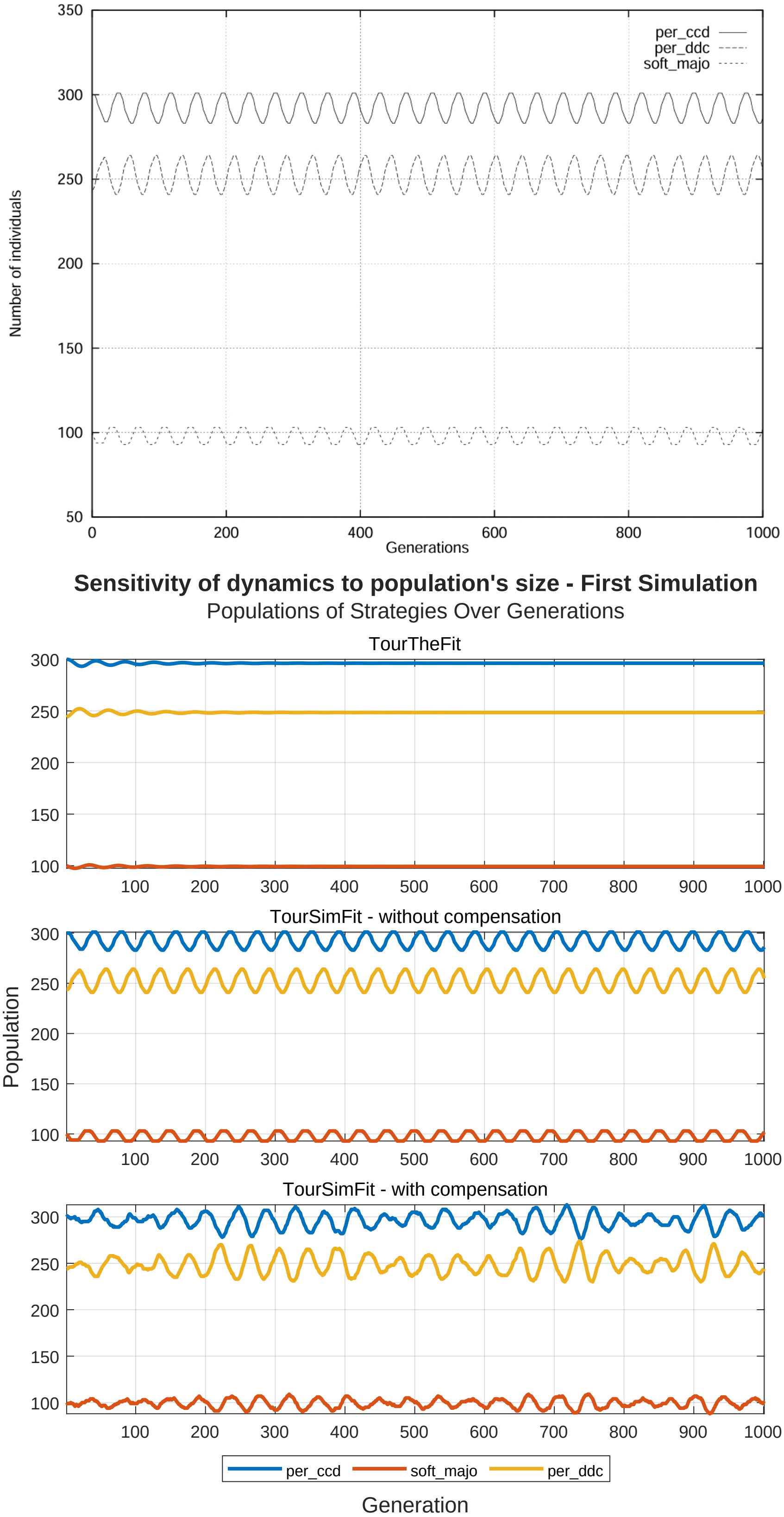


Figure 7: 7th Simulation - Sensitivity of dynamics to population's size - First Simulation

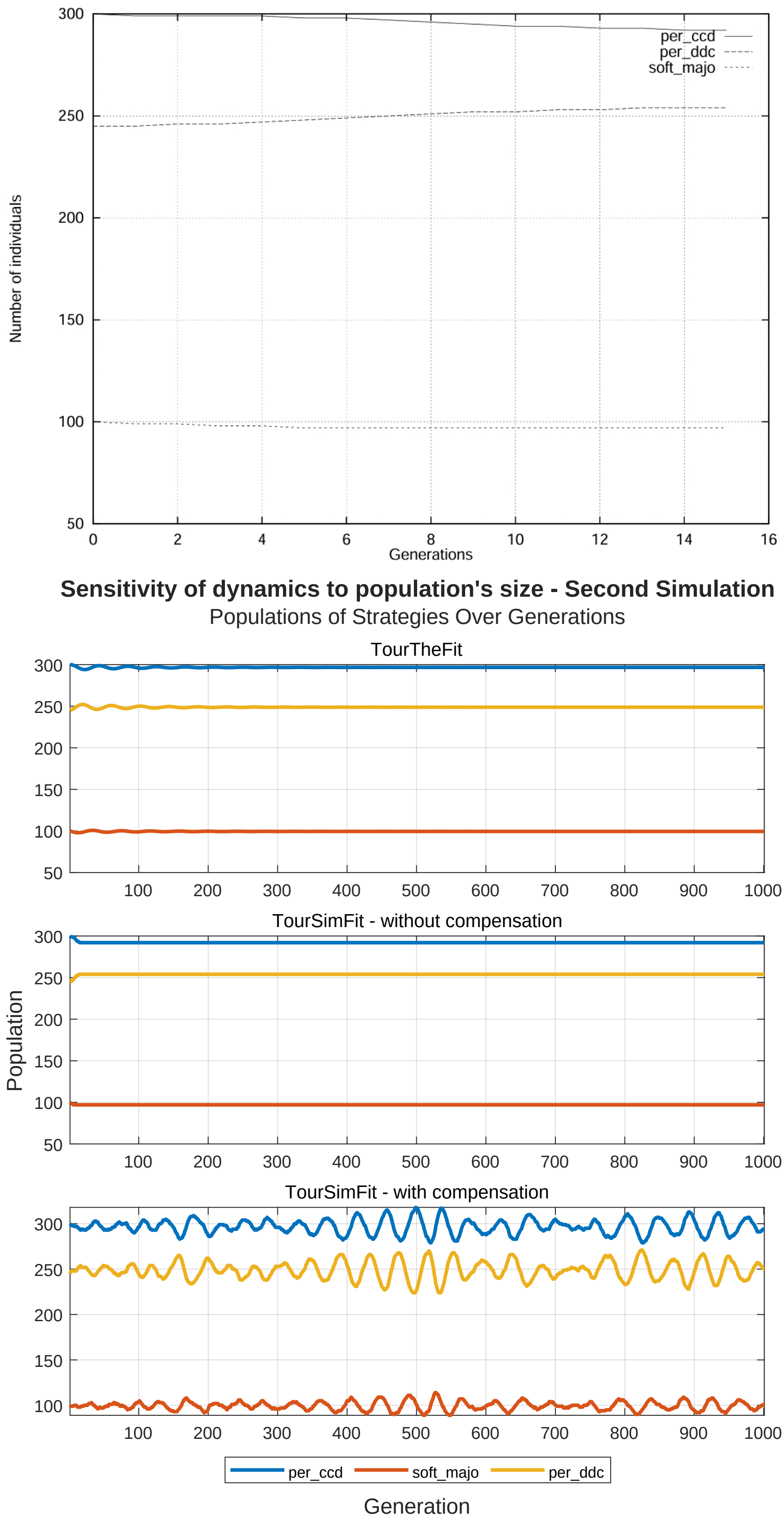


Figure 8: 8th Simulation - Sensitivity of dynamics to population's size - Second Simulation

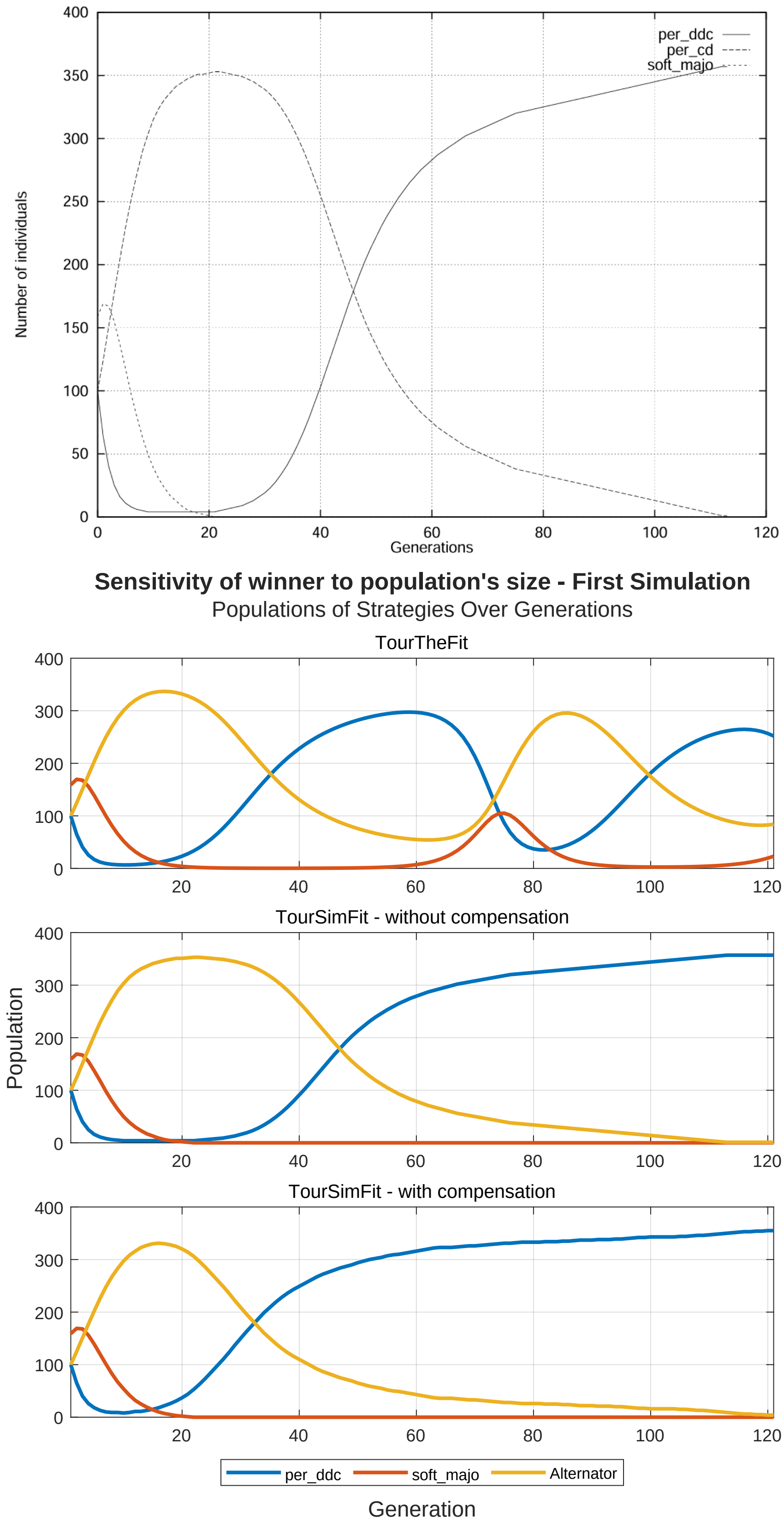
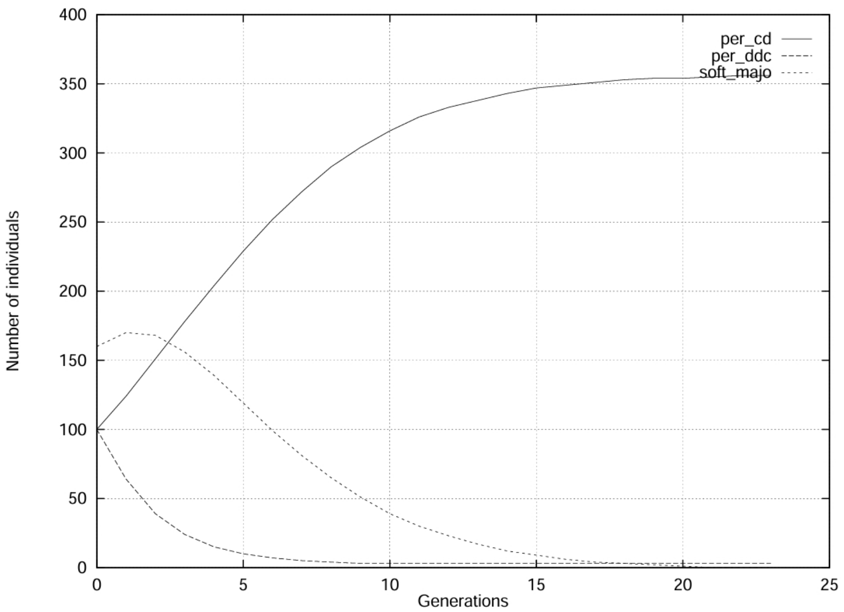


Figure 9: 9th Simulation - Sensitivity of winner to population's size - First Simulation



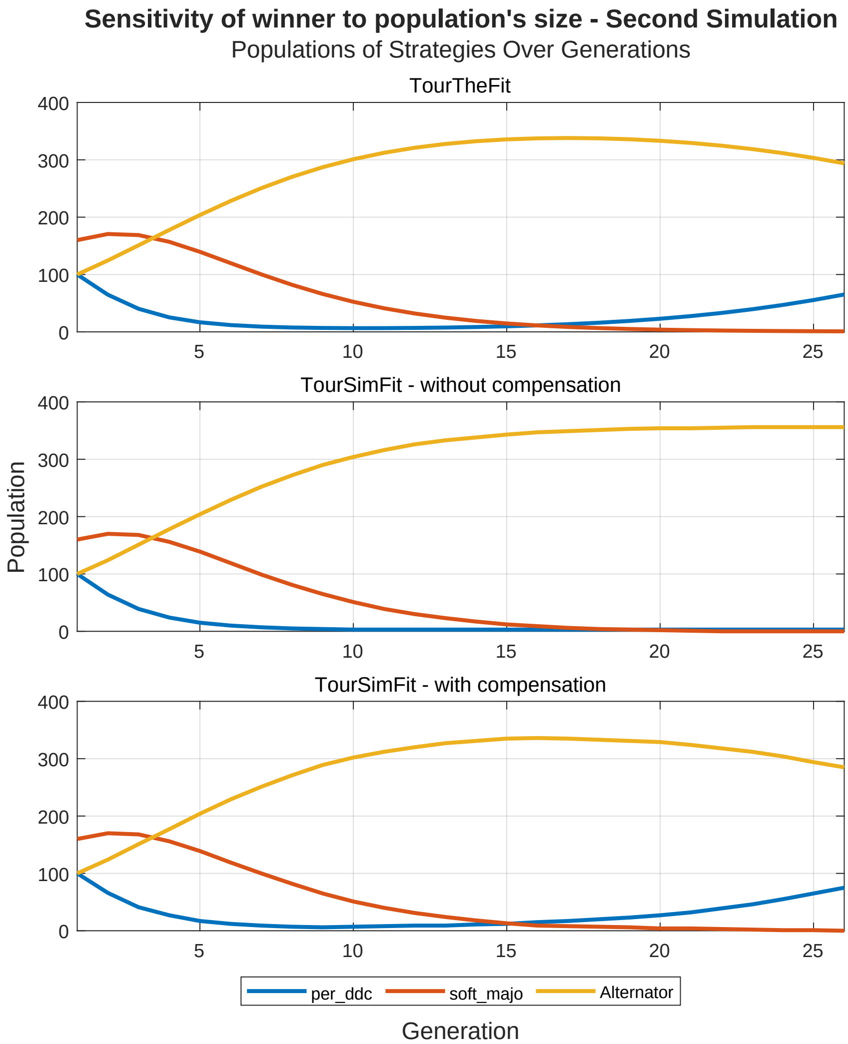
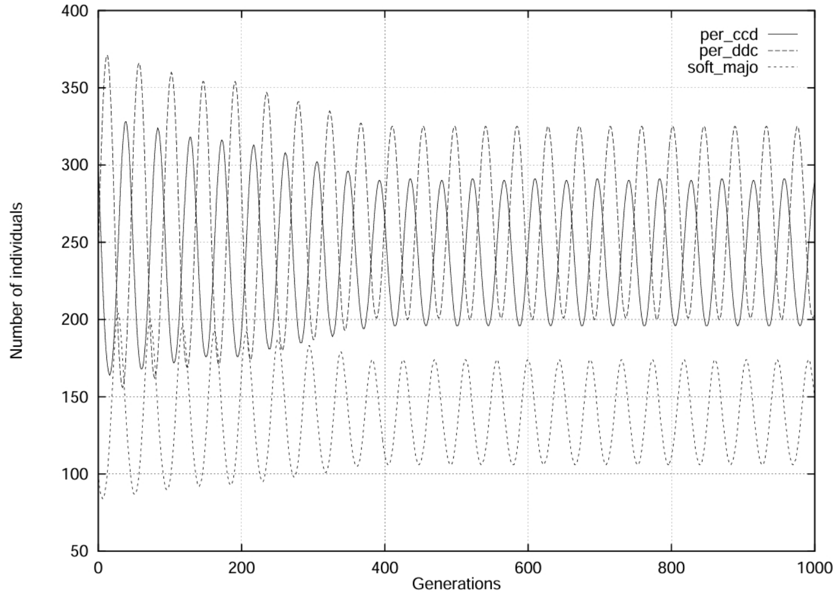


Figure 10: 10th Simulation - Sensitivity of winner to population's size - Second Simulation



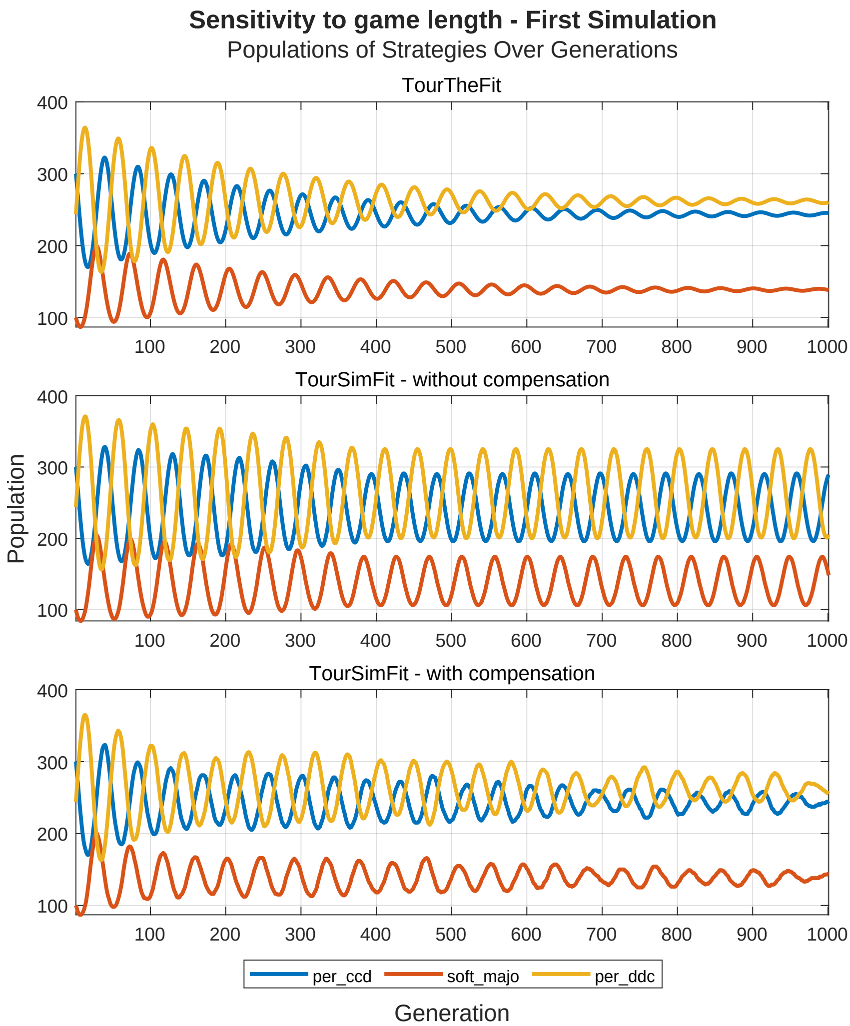


Figure 11: 11th Simulation - Sensitivity to game length - First Simulation

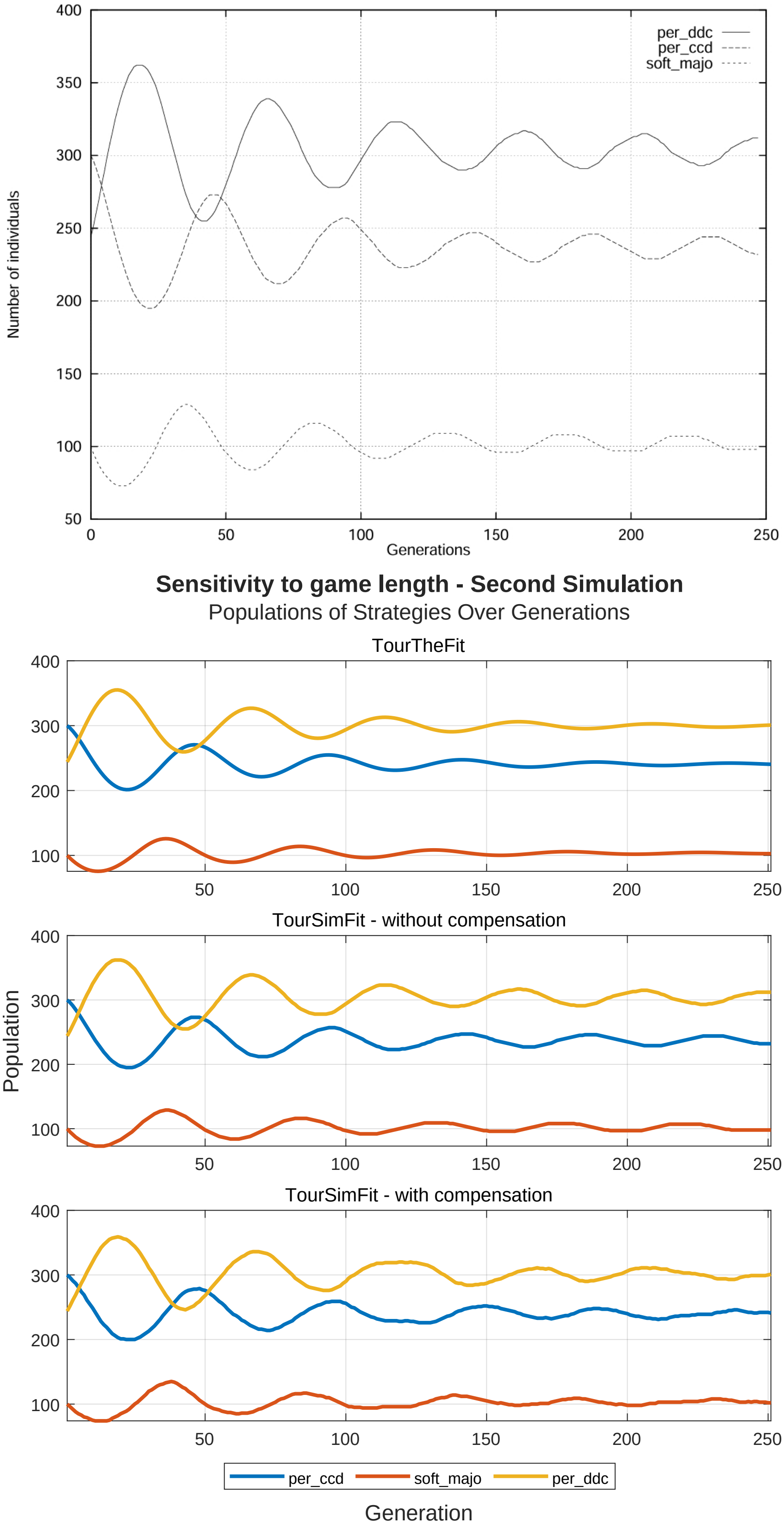


Figure 12: 12th Simulation - Sensitivity to game length - Second Simulation

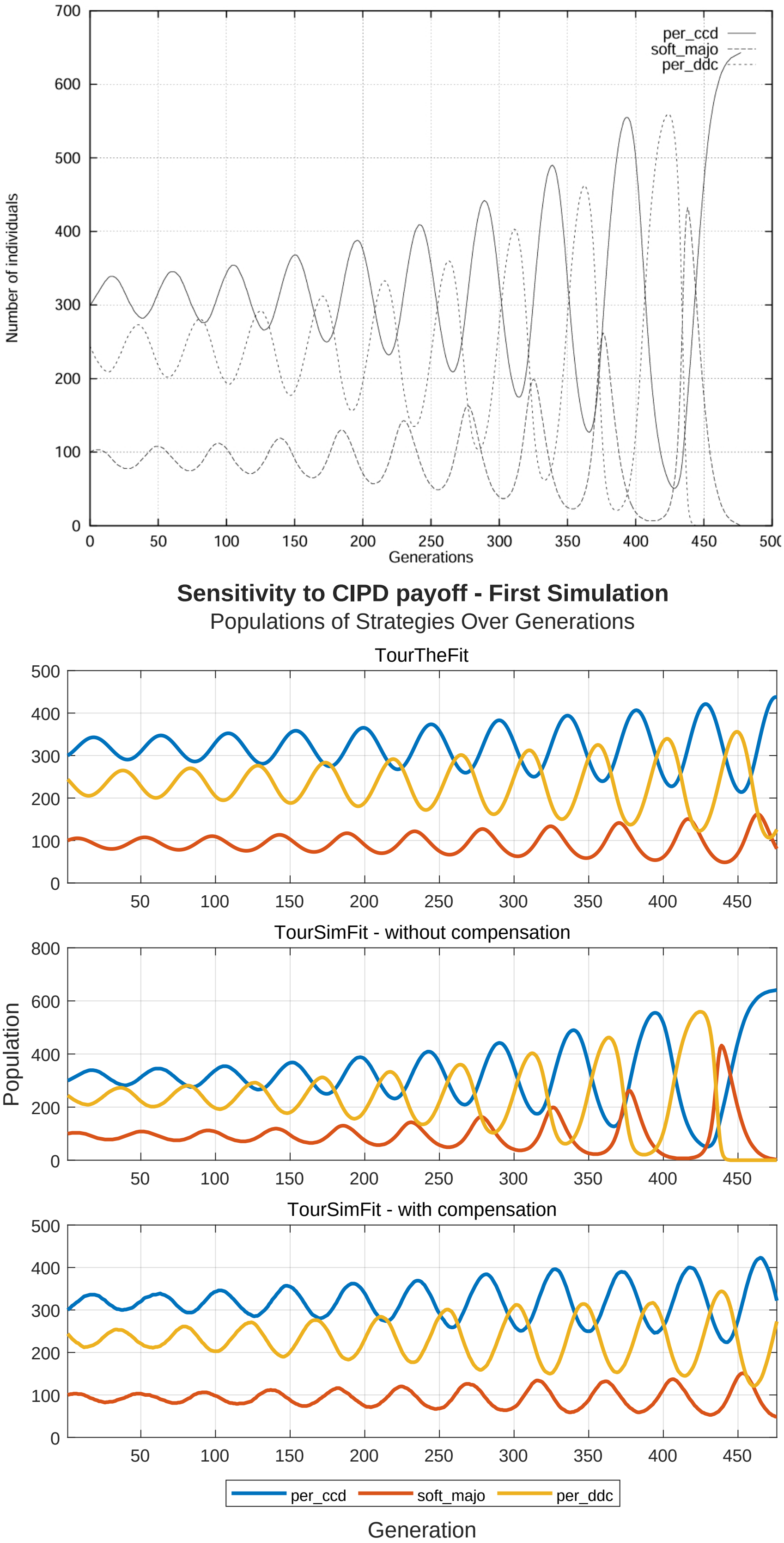


Figure 13: 13th Simulation - Sensitivity to CIPD payoff - First Simulation

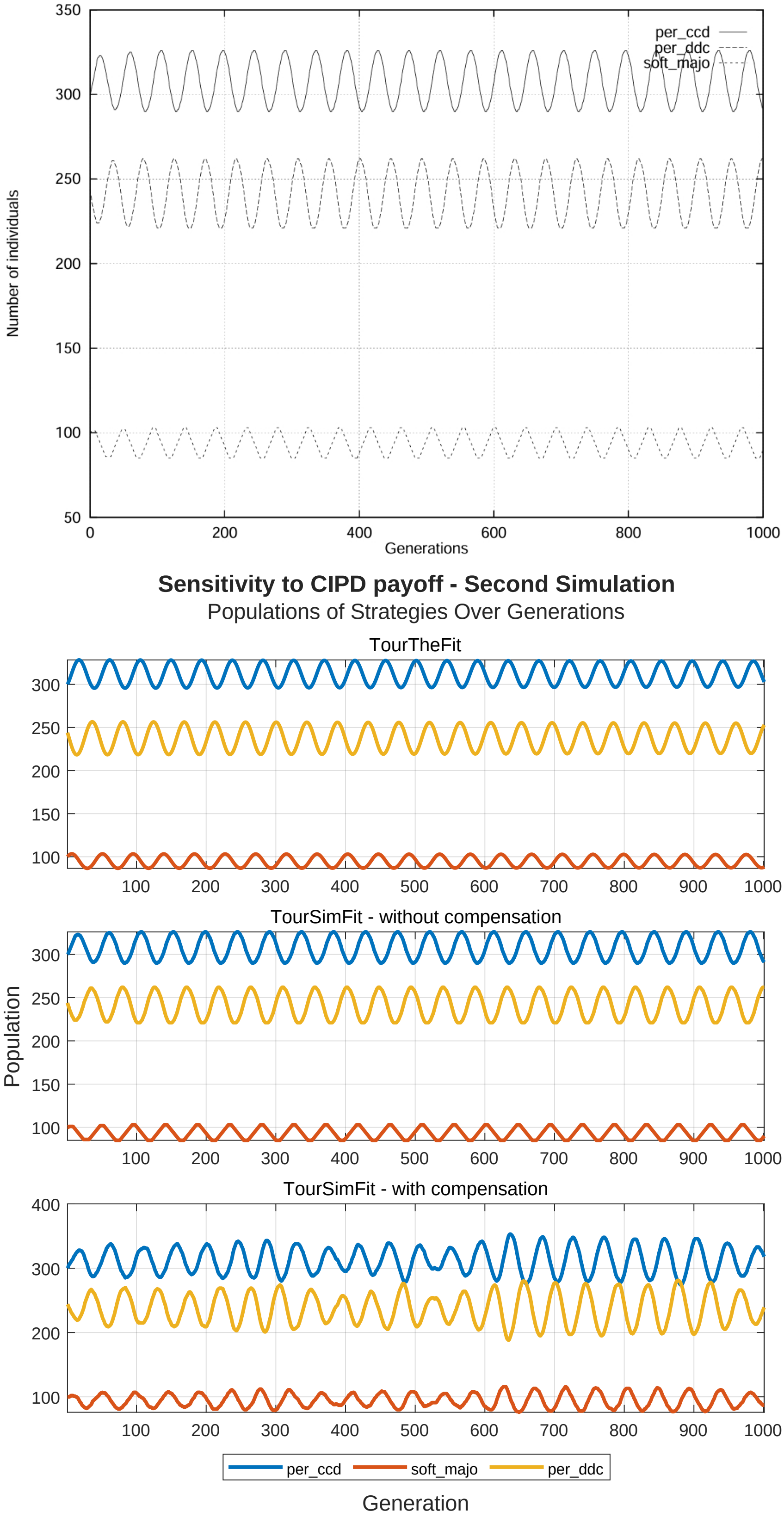


Figure 14: 14th Simulation - Sensitivity to CIPD payoff - Second Simulation

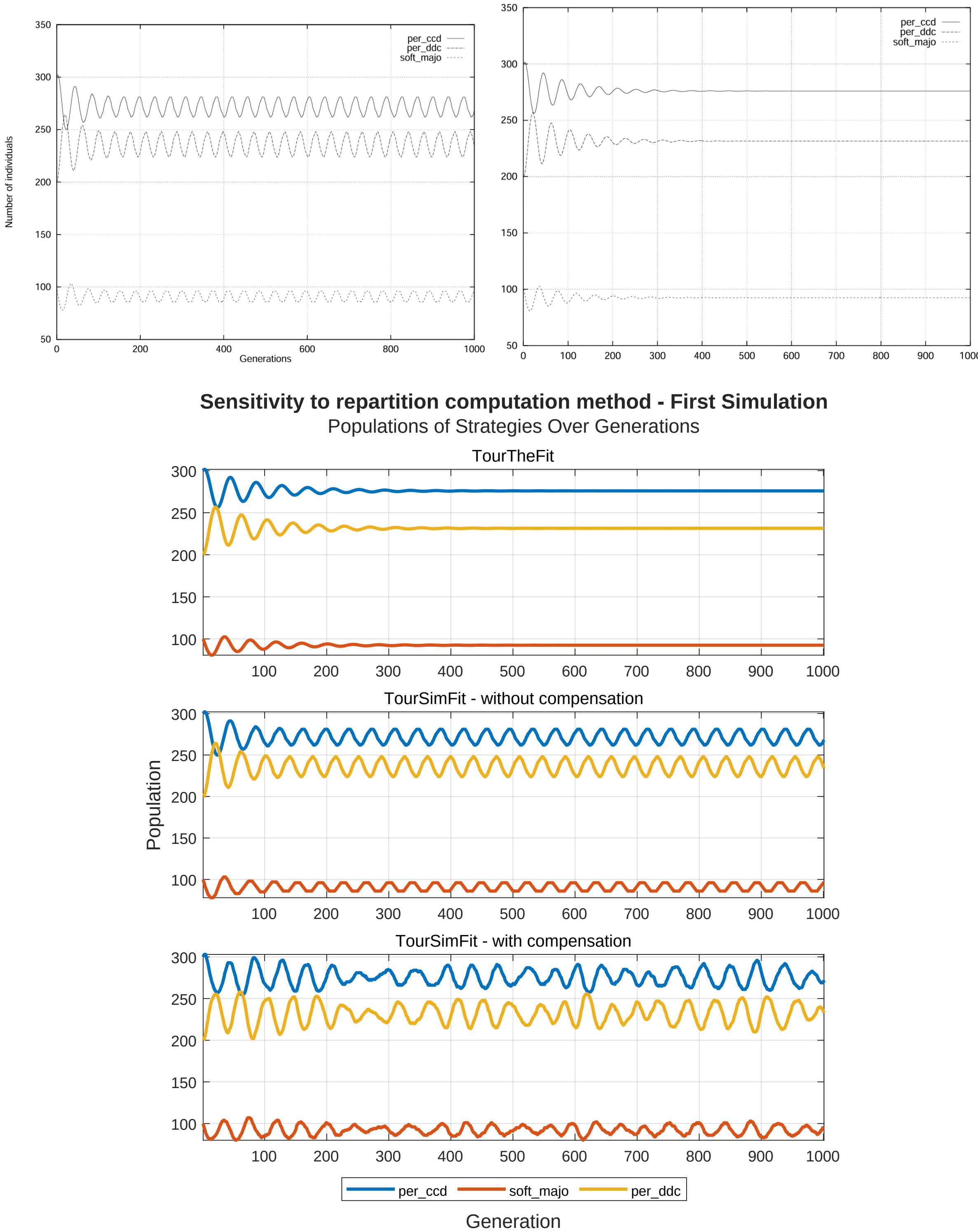


Figure 15: 15th Simulation - Sensitivity to repartition computation method - First Simulation

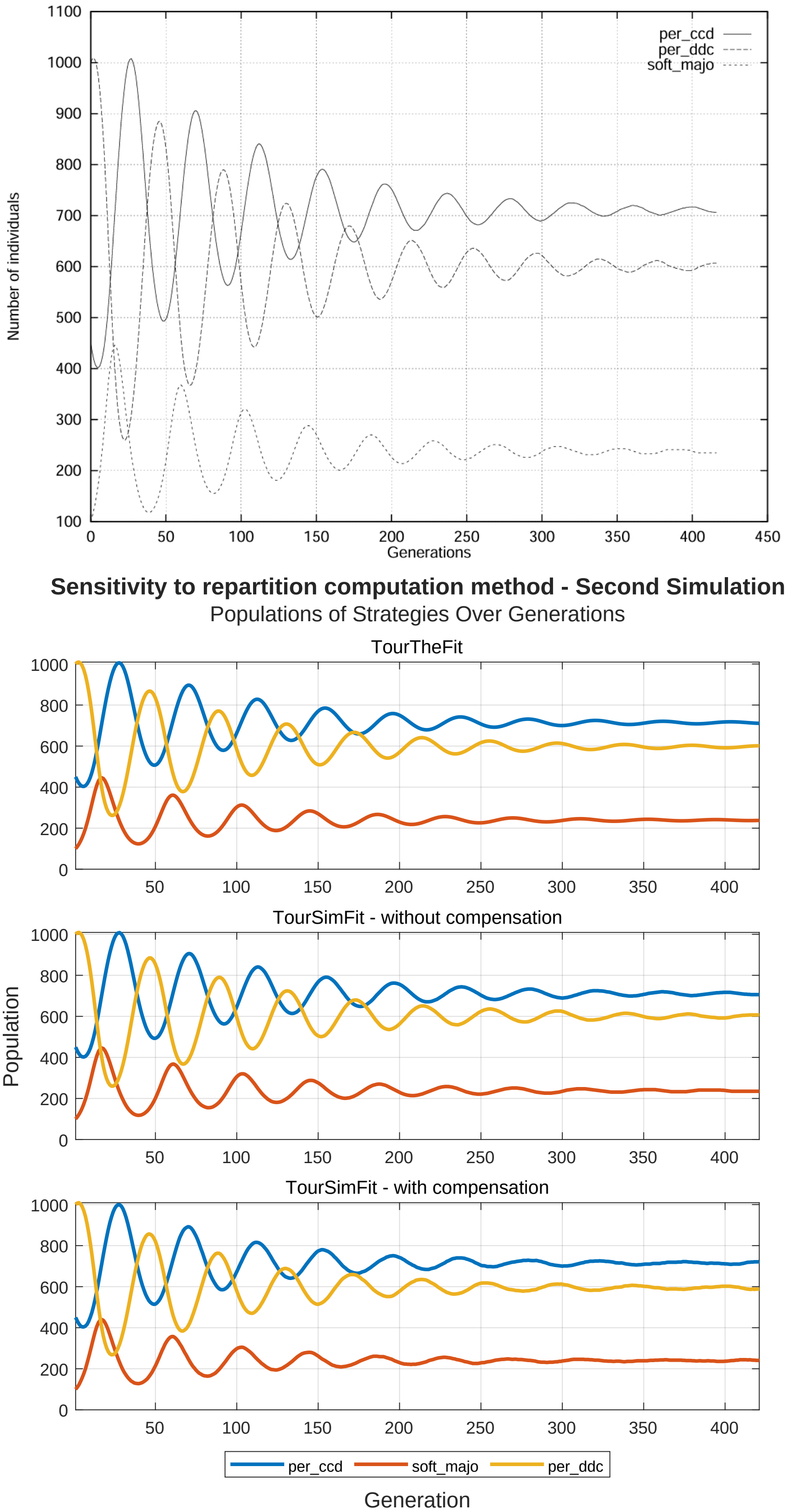


Figure 16: 16th Simulation - Sensitivity to repartition computation method - Second Simulation

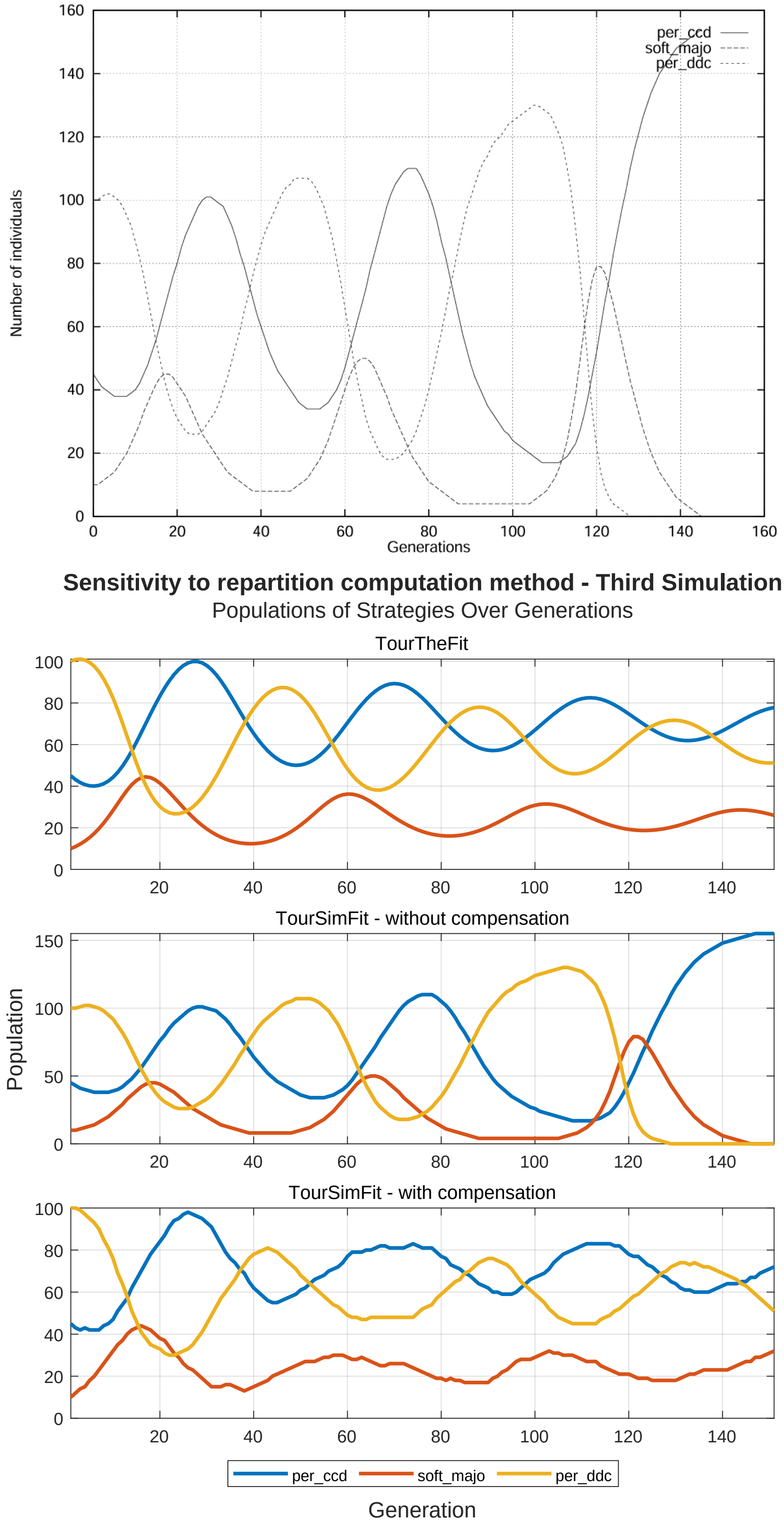


Figure 17: 17th Simulation - Sensitivity to repartition computation method - Third Simulation

4 Imitation Dynamics

The second evolutionary dynamic to be presented is that of Imitation Dynamics. In this case, the new distribution of the population is not calculated strictly as a function of the fitness of each strategy for each generation. Instead, a simpler and perhaps more realistic logic is adopted: in each generation, we identify the strategy (or, in case of a tie, the strategies) that performed best. Then, given a defined number K of players who change strategy per generation, K non-optimal strategy players are randomly selected and each adopts one of the optimal strategies, again randomly, in the case of ties.

It is important to highlight that the selection is only made among players who are not already using an optimal strategy, and not among all players in the population. We consider this to reflect reality more accurately — if a player is already following an optimal strategy, why would they consider changing? This choice has some consequences on the results that are presented below.

To determine the optimal strategy, a score calculation similar to that used in the fitness dynamics case is employed, in order to save computational time. That is, matches between every player are not actually simulated; instead, scores are calculated based on the payoffs of each strategy against each other strategy and the corresponding population sizes.

As will be shown below during the theoretical presentation of the TourTheImi function, this process can be formulated using a Markov chain, where the states are the various possible population distributions, based on the sum of the initial populations (the total population) and the number of strategies involved in the process. The theoretical background is crucial to understand the following implementations: initially, we consider the r-state Markov chain, where each state represents the individual strategy of each player. For example, for $N = 5$ players and 3 strategies, a possible r-state is (11323). We then consider the s-states, where a state represents the population of each strategy; in the previous example, the s-state would be (212). Essentially, this involves grouping various r-states into one s-state, which, since we are not truly interested in each individual player but in the total population per strategy, is equivalent. It is shown that the process $r(t)$ is lumpable and therefore that $s(t)$ is a Markov chain. This theoretical knowledge is necessary for the implementation of TourTheImi below.

4.1 The function TourSimImi

The function that simulates the evolutionary tournament with imitation dynamics is implemented as

$$[\mathrm{POP}, \mathrm{BST}] = \mathrm{TourSimImi}(B, \mathrm{Strategies}, \mathrm{POP}_0, K, T, J, \mathrm{mode}).$$

The inputs and outputs of the function are similar to those of TourSimFit, with the additional argument K, an integer value that specifies the number of players who change strategy per generation. The additional argument mode (the function runs with default value "Individual") can take the values "Individual" and "Total", and refers to the way in which the optimal strategy is selected: in the "Individual" case, the strategy of the best individual player is returned, while in the "Total" case, the strategy with the highest total score among the players using it is selected. The choice of mode yields significantly different results, as will be shown below. This is a pseudocode representation (3) of the updatePopulation subfunction, that updates the population after each generation.

4.2 The function TourTheImi

The TourTheImi function has the form

$$P = \text{TourTheImi}(B, \text{Strategies}, \text{POP}_0, K, T, J, \text{mode})$$

with arguments identical to the function TourSimImi, and output the transition matrix P of the Markov chain of the s-states. In reality, the initial population is used only to determine the total population size of the tournament (so it could simply be replaced by an argument N), and the number of generations J is not used at all.

For this analysis, we consider the s-states of the tournament as the number of players using each strategy. For example, for $N = 9$, we may have

$$s_1 = \begin{bmatrix} 0 & 0 & 9 \end{bmatrix}, \quad s_2 = \begin{bmatrix} 0 & 1 & 8 \end{bmatrix},$$

and so on. Depending on the mode argument, we expect the transition matrix to take different forms. For example, in "Individual" mode and with strategies All_D, All_C, and TitForTat, we expect the state

$$\begin{bmatrix} 0 & 5 & 4 \end{bmatrix}$$

to be absorbing, since there are no non-optimal players, and therefore it should have only one transition — to itself — with probability 1. In contrast, in the "Total" mode, this state would not be absorbing, as the All_C strategy accumulates more total points. Below is the pseudocode (4) for enumerating the possible transitions from each state.

Another function is also created:

$$\text{AnalyzeMarkovChain}(P, \text{POP}_0, \text{Strategies}, \text{Title}),$$

which is responsible for generating the state transition diagrams shown below.

Algorithm 3 Update Population Based on Imitation Dynamics

```
function updatePopulation(currentPop, bestStrategyIndices, K, numStrategies)

numBestStrategies ← length(bestStrategyIndices)
nonBestStrategyIndices ← set difference of {1, . . . , numStrategies} and best-
  StrategyIndices
nonBestPopulation ← sum of currentPop at nonBestStrategyIndices
nextPop ← currentPop
if nonBestPopulation = 0 then
    return
end if
actualK ← min(K, nonBestPopulation)
if actualK = 0 then
    return
end if
agentPool ← empty list
for each i in nonBestStrategyIndices do
    if currentPop[i] > 0 then
        Append i to agentPool currentPop[i] times
    end if
end for
if agentPool is not empty then
    Shuffle agentPool randomly
    selectedAgents ← first actualK elements of agentPool
    selectedCounts ← zero vector of length numStrategies
    for each i in selectedAgents do
        selectedCounts[i] ← selectedCounts[i] +1
    end for
    for each i in nonBestStrategyIndices do
        if selectedCounts[i] > 0 then
            nextPop[i] ← nextPop[i] − selectedCounts[i]
            for j = 1 to selectedCounts[i] do
                bestIdx ← random element from bestStrategyIndices
                nextPop[bestIdx] ← nextPop[bestIdx] +1
            end for
        end if
    end for
end if
return nextPop
```

Algorithm 4 Enumerate Transitions by Choosing actualK Agents from Non-Best Strategies

```
agentPool ← empty list
for each i in nonBestStrats do
   Append i to agentPool currPop[i] times
end for
combos ← all combinations of actualK indices from 1 to length(agentPool)
for c = 1 to number of rows in combos do
   selected ← agentPool at indices combos[c]
   nextPop ← currPop
   for idx = 1 to length of selected do
      from ← selected[idx]
      to ← random element from bestStrats
      nextPop[from] ← nextPop[from] −1
      nextPop[to] ← nextPop[to] +1
   end for
   key ← string representation of nextPop
   if key exists in transitionCounts then
      transitionCounts[key] ← transitionCounts[key] +1
   else
      transitionCounts[key] ← 1
   end if
end for
```

This function, based on the matrix P calculated by the TourTheImi function and the initial population POP0, classifies the states into transient, absorbing, and the initial state. It also further distinguishes between reachable and unreachable states based on the initial state. It generates a diagram of each state, where the state type is indicated with color, the population of each strategy is shown, the strategy names are labeled (as given by the Strategies argument), and the corresponding transition probabilities are displayed above each transition. The Title argument is the title of the resulting diagram.

Another function for the generation of state transition graphs is the following:

$$\mathrm{PlotStateTransitionGraph}(P, \mathrm{POP}_0, \text{Strategies}, \text{Title}),$$

which is responsible for visualizing the state transition dynamics of the Markov process. This function takes as input the transition matrix P, the initial population vector POP0, and a cell array of strategy names (Strategies) to generate a directed graph that represents the transitions between population states.

The full state space is constructed using the total population and number of strategies, and the function computes the long-run behavior of the system by raising the transition matrix to a high power (P^{100}). Specific reference states, such as full adoption of individual strategies, are identified, and the long-run probabilities of reaching these states from all other states are extracted and normalized.

These normalized values are then used to color the nodes in the graph, indicating their asymptotic convergence tendencies. The graph layout is based on a 2D projection of the state space, where the x-axis and y-axis represent the populations of specific strategies (e.g., All_D and Grim), allowing for intuitive interpretation of the state distribution.

The graph is created using MATLAB's digraph object, with self-loops removed to improve clarity. The resulting figure is formatted to fit within an A4 landscape layout and is exported as a vector-based PDF using the specified Title as the filename.

4.3 Simulations - Examples

Below are some simulations generated by the functions described above, which are of interest. Each simulation is located in a specific example file in the Examples folder and by running the files properly, the exact results presented below are produced.

4.3.1 1st Simulation - Example usage of TourTheImi, AnalyzeMarkovChain and TourSimImi

In the first simulation 18, the Markov chain resulting from the strategies $\begin{bmatrix} \mathit{All\_D} & \mathit{All\_C} & \mathit{TitForTat} \end{bmatrix}$ in the mixture $\begin{bmatrix} 1 & 5 & 3 \end{bmatrix}$ is presented.

From the resulting diagram, we observe that the possible absorbing states are $\begin{bmatrix} 9 & 0 & 0 \end{bmatrix}$, which corresponds to the dominance of the *All_D* strategy, $\begin{bmatrix} 0 & 0 & 9 \end{bmatrix}$, corresponding to the dominance of the *TitForTat* strategy, and $\begin{bmatrix} 0 & 1 & 8 \end{bmatrix}$, corresponding to the dominance of *TitForTat* with the survival of *All_C*. From the transition probabilities, shown on the transition arrows, we observe that the last absorbing state has a significantly lower probability of occurring compared to the other two.

Furthermore, we observe that the states in which only the cooperative strategies exist (*All_C* and *TitForTat*), i.e., strategies that never defect first and therefore their players achieve equal scores when playing against each other, transition only to themselves, since there are no players with a lower score than the maximum.

In Figure 19, two executions of the TourSimImi for the strategy mixture of the above simulation are illustrated. We observe that due to the random selection of imitators—i.e., players who do not have the best strategy and switch to one of the better ones—the final mixture of strategies differs for each execution of the program.

Specifically, in execution (a), we observe

$$\begin{bmatrix} 1 & 5 & 3 \end{bmatrix} \to^* \begin{bmatrix} 9 & 0 & 0 \end{bmatrix}$$

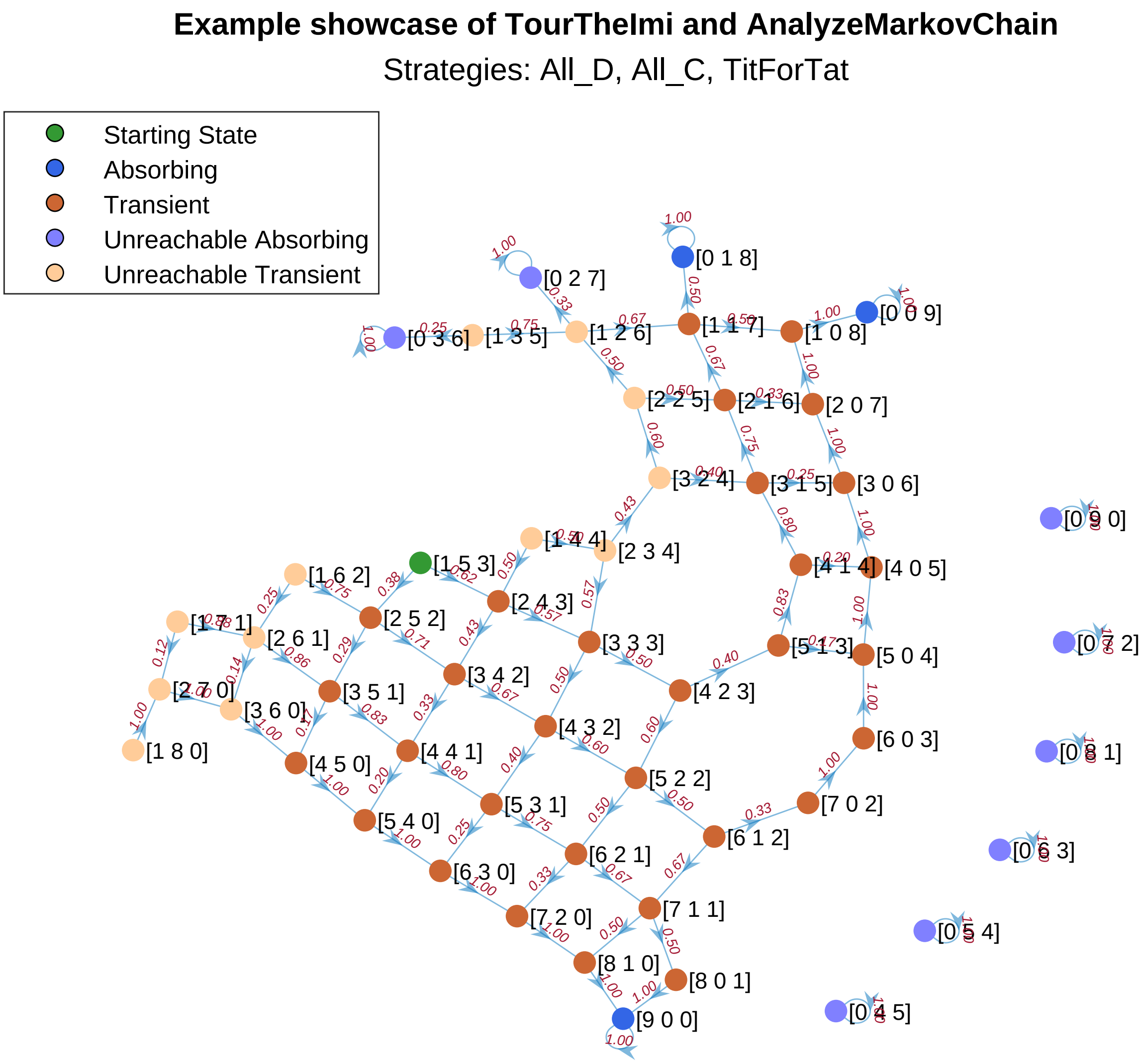


Figure 18: Example showcase of TourTheImi and AnalyzeMarkovChain

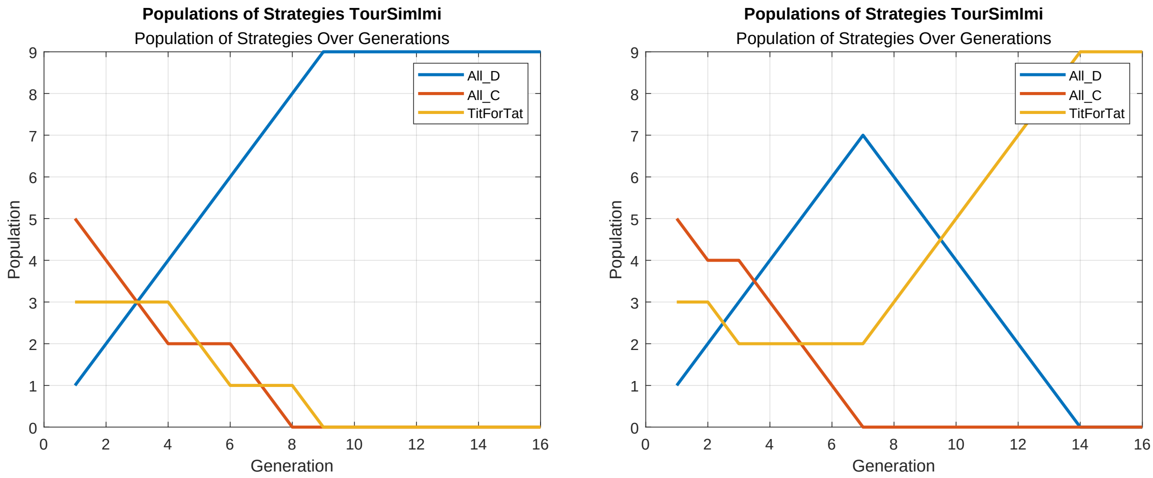
while in execution (b),

$$\begin{bmatrix}1 & 5 & 3\end{bmatrix} \rightarrow^* \begin{bmatrix}0 & 0 & 9\end{bmatrix},$$

which, as we analyzed above, are indeed the two most likely absorbing states. Run example34 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.2 2nd Simulation - Default mode (Individual) trial for best strategy calculation

In the next two simulations, an attempt is made to demonstrate the difference in results caused by the different methodologies used to select the best strategy. Initially (Figure 20), the Markov chain resulting from the initial strategy mixture $\begin{bmatrix}1 & 4 & 5\end{bmatrix}$ and the "Individual" selection method is presented, where the best strategy is chosen by comparing the payoffs of the strategies in one-on-one games between each pair of strategies.



(a) $\begin{bmatrix}9 & 0 & 0\end{bmatrix}$ (b) $\begin{bmatrix}0 & 0 & 9\end{bmatrix}$

Figure 19: Absorbing States may differ even for the same Starting State

The result is similar to that of the first simulation 18. With some observation, we can envision an imaginary curve connecting the vertices of the states $\begin{bmatrix}8 & 0 & 2\end{bmatrix}, \begin{bmatrix}6 & 1 & 3\end{bmatrix}, \begin{bmatrix}4 & 2 & 4\end{bmatrix}, \begin{bmatrix}2 & 3 & 5\end{bmatrix}$, which separates the basins of attraction of the cooperative and non-cooperative strategies.

If the system is in a state below this curve, it will end up in the absorbing state where “All_D” dominates. Conversely, if it starts in a state above this curve, it will end up in one of the absorbing states of the cooperative strategies. Run example35 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.3 3rd Simulation - “Total” mode trial for best strategy calculation

In the third simulation (Figure 21), with the same initial strategy mixture $\begin{bmatrix}1 & 4 & 5\end{bmatrix}$ and using the “Total” method—i.e., comparing the players’ payoffs in each state (as computed by Axelrod) using the populations of the state, and selecting as the best strategy the one belonging to the player with the highest total payoff—we observe different behavior.

In contrast to the previous simulation (Figure 20), here we observe that the diagram is divided into three distinct subgraphs.

The first subgraph, which includes the initial state, is the only one that contains a reachable absorbing state ($\begin{bmatrix}0 & 0 & 10\end{bmatrix}$), corresponding to the dominance of “TitForTat”. This occurs because, under the “Total” mode for selecting the best strategy, the cooperative strategies collaborate with each other and, due to their greater numbers, overcome the advantage that defection gives to “All_D”.

The second subgraph includes the absorbing state where “All_D” dominates ($\begin{bmatrix}10 & 0 & 0\end{bmatrix}$), but this state, as well as all the transitional states in this subgraph, are unreachable for the reason described above.

Finally, the third subgraph includes the absorbing state where "All_C" dominates ($\begin{bmatrix} 0 & 10 & 0 \end{bmatrix}$), which is also unreachable, as are all the transitional states in this subgraph, for the same reason. Run example36 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.4 4th Simulation - Sensitivity of Imitation Dynamics, mode = 'Individual', to population's size - Case N=3

For the next four simulations, the sensitivity of the imitation dynamics in the case of "Individual" mode to the total population's size is discussed. For all following simulations, the population consists of members of the following three strategies: All_C, All_D and Grim (which in this case is an equivalent strategy to TitForTat). Another important aspect to discuss is the form of the plots produced; a new function PlotStateTransitionGraph is used in order to plot the possible states of the Markov Chain in a triangular form. More specifically, the x-axis of the plot refers to the population of All_D, the y-axis represents the population of Grim and, since the total population N remains constant, the population of All_C increases as we move towards the bottom-left of the plot.

The analysis begins with the simple case of $N = 3$. This is the proper instance to comment on the colors of the vertices; the color is produced by calculating the final state probability matrix (by raising the calculated matrix P to a sufficiently large power). The more red a vertex appears, the more likely this population distribution is to lead to the domination of All_D. The more blue it appears, the more likely it is to lead to the domination of Grim. Lastly, the more green it is, the more likely it is to lead to All_C domination. You may also notice a few black vertices; these are caused by the fact that for some population distributions (as we also saw earlier) the transition to total dominating states like the ones described above is impossible; with the current dynamics, for example, in a population of only All_C and Grim, no changes happen to the strategy distribution, meaning the state is also absorbing (if the simulation starts there, it ends there).

We begin to notice a trend in the case $N = 3$; only one vertex is green, the vertex where all 3 players follow the All_C strategy. States in the far left side of the plot are, as discussed earlier, black, except for the one on the top, which is blue (all 3 players follow the Grim strategy). Lastly, there is a distinction between states that lead to All_D domination and states that lead to Grim domination, with no states possibly leading to both; thus, in the case $N = 3$ the winning strategy is deterministic to the initial population. The result can be seen in Figure 22, subfigure (a). Run example18 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.5 5th Simulation - Sensitivity of Imitation Dynamics, mode = 'Individual', to population's size - Case N=5

In the next simulation, the total population is increased to 5. It is expected that the overall trend will continue, but with the increase in population we at some point expect to see states that may lead to both All_D domination and states that lead to Grim domination. That is precisely the case; as we can see by the colors of the vertices and by the edges added to the graph, states 9 and 13 both have this property. It should also be noted that state 8 may possibly lead to state 7 as well as the expected state 1, the domination of Grim state. This happens because the non-best strategies of this state are All_D and All_C; using $K = 1$, as in this example, it is random whether the defector or the cooperator switch strategies first, and thus it is random whether the Grim strategy totally dominates. The result can be seen in Figure 22, subfigure (b). Run example19 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.6 6th Simulation - Sensitivity of Imitation Dynamics, mode = 'Individual', to population's size - Case N=10

As the population keeps increasing, as in the case of this simulation, where the total population is increased to 10, we expect the results to change as before; more states are going to possibly lead to both All_D domination and states that lead to Grim domination. Also, the higher and to the right of the plot, the more likely Grim domination is to occur. The expected results are the ones also reproduced in the Figure. The result can be seen in Figure 22, subfigure (c). Run example20 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.7 7th Simulation - Sensitivity of Imitation Dynamics, mode = 'Individual', to population's size - Case N=100

Lastly, one final large increase to the population to $N = 100$ is made, which aims to showcase a more continuous form of the graph; in this case, the exact possible transitions of each state are not visible, however the colors of the vertices "paint" (pun intended) a complete picture of the situation. More specifically, it is made clear where the states that lead to each possible final state reside inside the graph. This generalizes the concept presented thus far; a red triangle in the bottom left of states that lead to All_D domination, a blue triangle in the top right of states that lead to Grim domination, a purple line of states possibly leading to both separating the two, a green vertex in the bottom left representing the All_C domination (from the start) and a black line in the left representing possible absorbing states different from the ones with a single strategy alive. The result can be seen in Figure 22, subfigure (d). Run example21 of the Examples folder (after reading Quickstart guide) to recreate the figure. Also note that this specific example, because of the lack of optimization, is extremely slow to run.

4.3.8 8th Simulation - Sensitivity of Imitation Dynamics, mode = 'Individual', to payoff matrix - Case a=3.2

For all previous 4 simulations, the payoff matrix was regarded to be constant and equal to $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$. As discussed during the analysis of the Fitness Dynamics, the payoff matrix can drastically change the results of a simulation. Thus, for the next 4 simulations, the total population is kept constant $N = 10$ and the payoff matrix is changed with the forms $B = \begin{bmatrix} a & 1 \\ 4 & 2 \end{bmatrix}$, with $a \in [3, 4)$. This means that the payoff of the cooperation-cooperation result will be increased and the results will be observed. It is expected that such an increase will eventually lead to more strategies leading to the domination of the nice strategies (i.e. Grim domination).

The analysis begins with the case $a = 3.2$. The result, as expected, is similar to the case of $a = 3$ which was indirectly presented in the 6th Simulation, with only a few more strategies leading to Grim domination. The result can be seen in Figure 23, subfigure (a). Run example22 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.9 9th Simulation - Sensitivity of Imitation Dynamics, mode = 'Individual', to payoff matrix - Case a=3.4

The payoff is increased to $a = 3.4$, with the overall effect of the increase becoming more visible. The blue triangle in the top right is expanding towards the bottom left, meaning that even in the cases with a lot of initial All_C members, because the cooperation payoff is increased a lot, it is easier for the Grim strategy to become the most viable. Thus, the general trend is for the blue triangle to approach the bottom left of the plot. The result can be seen in Figure 23, subfigure (b). Run example23 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.10 10th Simulation - Sensitivity of Imitation Dynamics, mode = 'Individual', to payoff matrix - Case a=3.6

The further increase to $a = 3.6$ further increases the blue vertices in the way discussed above. The result can be seen in Figure 23, subfigure (c). Run example24 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.11 11th Simulation - Sensitivity of Imitation Dynamics, mode = 'Individual', to payoff matrix - Case a=3.8

Lastly, with the increase to $a = 3.8$, the blue triangle becomes a right triangle, meaning the only red states are the ones in the complete bottom of the plot, no longer forming a triangle. It should also be noted that there are no more purple states; the result once again becomes deterministic. It should come as no surprise that there still exist red states; the first line represents an initial population of only All_C and All_D, clearly making All_D the better strategy and leading to red coloring. The line above that corresponds to a single Grim participant in the initial population. The single Grim participant can only ever nearly tie with the All_D members, and each All_D member still gets more points from facing the All_C members than the participant of Grim does, thus again making All_D the best strategy. The result can be seen in Figure 23, subfigure (d). Run example25 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.12 12th Simulation - Sensitivity of Imitation Dynamics, mode = 'Total', to population's size - Case N=3

For the next simulations, the "Total" mode for best strategy calculation is selected. The same analysis as before will occur, meaning we will first see the effect of increasing the total population of the simulation. In the case of "Total", the total number of absorbing states depends on the total population as well as the strategies themselves. For example, in the case $N = 3$, if all 3 strategies used are nice, meaning there is never a defection, the state $[111]$ would be absorbing. Also, in the case we analyze with 2 nice strategy and a single naughty one, the state $[nn0]$ would be absorbing, meaning that if the total population is an even number, there is one extra absorbing state. Otherwise, the absorbing states are the states of domination by all strategies. In the specific case of $N = 3$, there is one extra absorbing state, the one with 1 person using All_D and 2 people using All_C; this is specific to the payoff matrix $\begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$. The All_D strategy gets $2 \cdot 4 \cdot T$ points, whereas the All_C strategy gets $2 \cdot (1 \cdot T + 3 \cdot T)$, where T is the number of rounds played per match, which are equal. Therefore, both strategies are best and the state is an absorbing one. Otherwise, the results are as expected: the plot is divided into areas starting from which domination of each strategy is eventually the result. The result can be seen in Figure 24, subfigure (a). Run example26 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.13 13th Simulation - Sensitivity of Imitation Dynamics, mode = 'Total', to population's size - Case N=5

The population is increased to 5. Except for possible special absorbing states like the one in the previous case, we expect the graph to be similar to the previous one, just with more states leading to each absorbing state. This is indeed the case, and there are not any extra absorbing states to be observed. The result can be seen in Figure 24, subfigure (b). Run example27 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.14 14th Simulation - Sensitivity of Imitation Dynamics, mode = 'Total', to population's size - Case N=10

The further increase makes the plot a little more interesting by adding the question: where does the area of each strategy end? Also notice state 46, which is the state with 5 All_C and 5 Grim, matching the example mentioned previously. Overall, however, the plot is as expected. The result can be seen in Figure 24, subfigure (c). Run example28 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.15 15th Simulation - Sensitivity of Imitation Dynamics, mode = 'Total', to population's size - Case N=100

Lastly, the population is increased to 100 to make the graph a "continuous" and to observe how each area of domination is defined. The areas for the specific payoff matrix has the form presented. The result can be seen in Figure 24, subfigure (d). Run example29 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.16 16th Simulation - Sensitivity of Imitation Dynamics, mode = 'Total', to payoff matrix - Case a=3.2

The last set of simulations presents the sensitivity to the payoff matrix, in terms of the "Total" mode for best strategy calculation. As in the case of the "Individual" mode, the payoff matrix has form $B = \begin{bmatrix} a & 1 \\ 4 & 2 \end{bmatrix}$, with $a \in [3, 4)$. As a is increased, it is expected that the blue and green areas (which represent the states that end in domination of a nice strategy, meaning strategies that tend to cooperate) will increase, whereas the red area corresponding to All_D domination will shrink. For this first increase to $a = 3.2$, the change is not visible. The result can be seen in Figure 25, subfigure (a). Run example30 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.17 17th Simulation - Sensitivity of Imitation Dynamics, mode = 'Total', to payoff matrix - Case a=3.4

For this simulation, a is increased even further to 3.4. The only noticeable change is state 56, which changes from red to green. The result can be seen in Figure 25, subfigure (b). Run example31 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.18 18th Simulation - Sensitivity of Imitation Dynamics, mode = 'Total', to payoff matrix - Case a=3.6

Next, a is increased to 3.6, again without any noticeable changes. The result can be seen in Figure 25, subfigure (c). Run example32 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.3.19 19th Simulation - Sensitivity of Imitation Dynamics, mode = 'Total', to payoff matrix - Case a=3.8

Lastly, a is changed to 3.8, with again very few specific changes, state 50 goes from red to green and state 17 changes from red to blue. In general, the expected trend is followed. However, it is followed to a much smaller degree than expected; throughout the change only 3 states ever changed color. Thus, we deduce that the total size of the population of each strategy is much more impactful on the end result than the payoff matrix, at least with the specific range of a. This happens because the best strategy is calculated by the total score of all people using that specific strategy, meaning that the number of people of the strategy is crucial. Overall, because of this element of the "Total" mode, along with the fact that we did not find any cases where the outcome is non-deterministic, we find it less interesting than the "Individual" mode. The result can be seen in Figure 25, subfigure (d). Run example33 of the Examples folder (after reading Quickstart guide) to recreate the figure.

4.4 Discussion

From the above, the following final conclusions emerge regarding the evolutionary Imitation Dynamics:

1. The choice of the specific dynamics followed when computing the populations of the next generation is crucial for the resulting outcomes. For example, selecting K players from the total population or K players from the non-optimal ones leads to very different results, both theoretically and in simulations.

2. The cases where the optimal strategy is computed using the Individual or Total methods also lead to drastically different outcomes regarding the

nature of the resulting states, as shown in Simulations 2 and 3. In general, the Total method creates cohesive subgraphs that have as terminal states the domination of one of the strategies, while the Individual method creates a larger subgraph that leads to many possible terminal states and some isolated terminal states. The term “cohesive” is used here loosely, as we are dealing with directed graphs in which there are generally no paths in both directions, but the term serves as an informal description of the phenomenon.

3. Due to point (2), we observe that the final state is not always predictable from the initial one when using the Individual method. This is because the random assignment in each generation may give different strategies an advantage in the next generation, leading to variability in the outcomes. In contrast, the Total method is fully deterministic with respect to the initial and final state, and only the path followed between them contains randomness.

4. In every case, the populations produced by the TourSimImi function follow a certain sequence of changes, which is represented in the state transition diagram generated by the corresponding TourTheImi and Analyze-MarkovChain or PlotStateTransitionGraph.

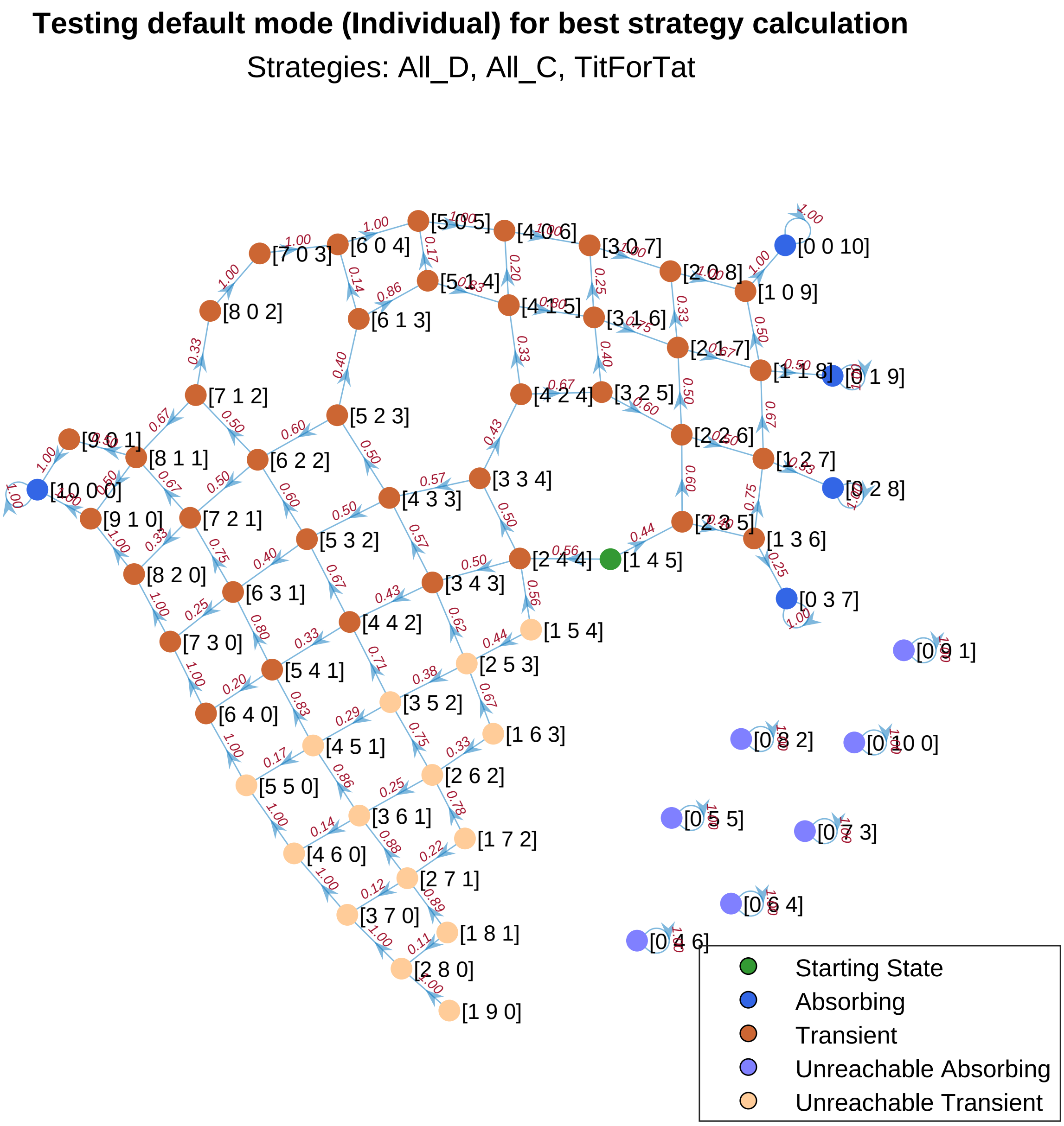


Figure 20: Testing default mode (“Individual”) for best strategy calculation with $POP0 = \begin{bmatrix} 1 & 4 & 5 \end{bmatrix}$

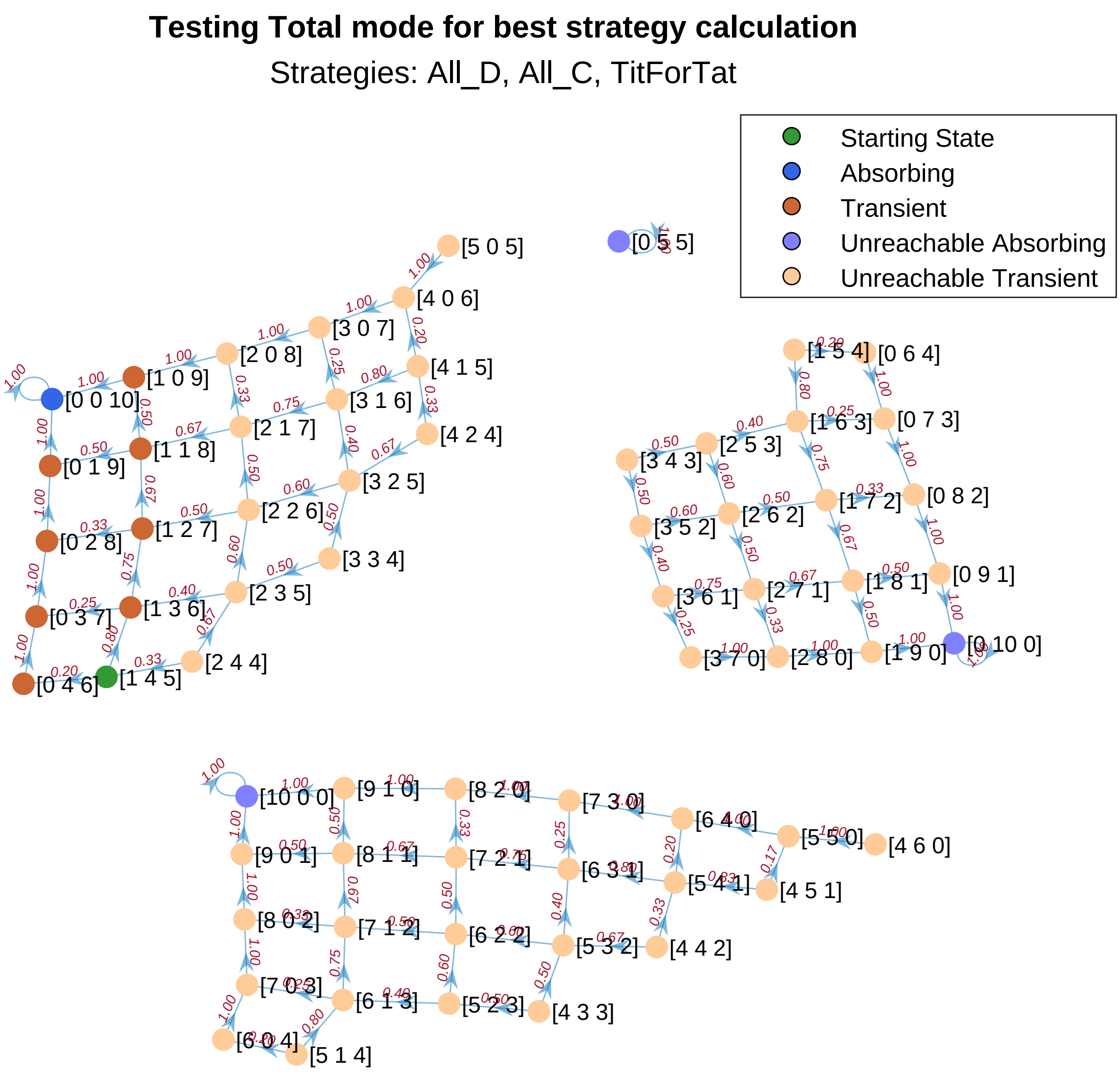
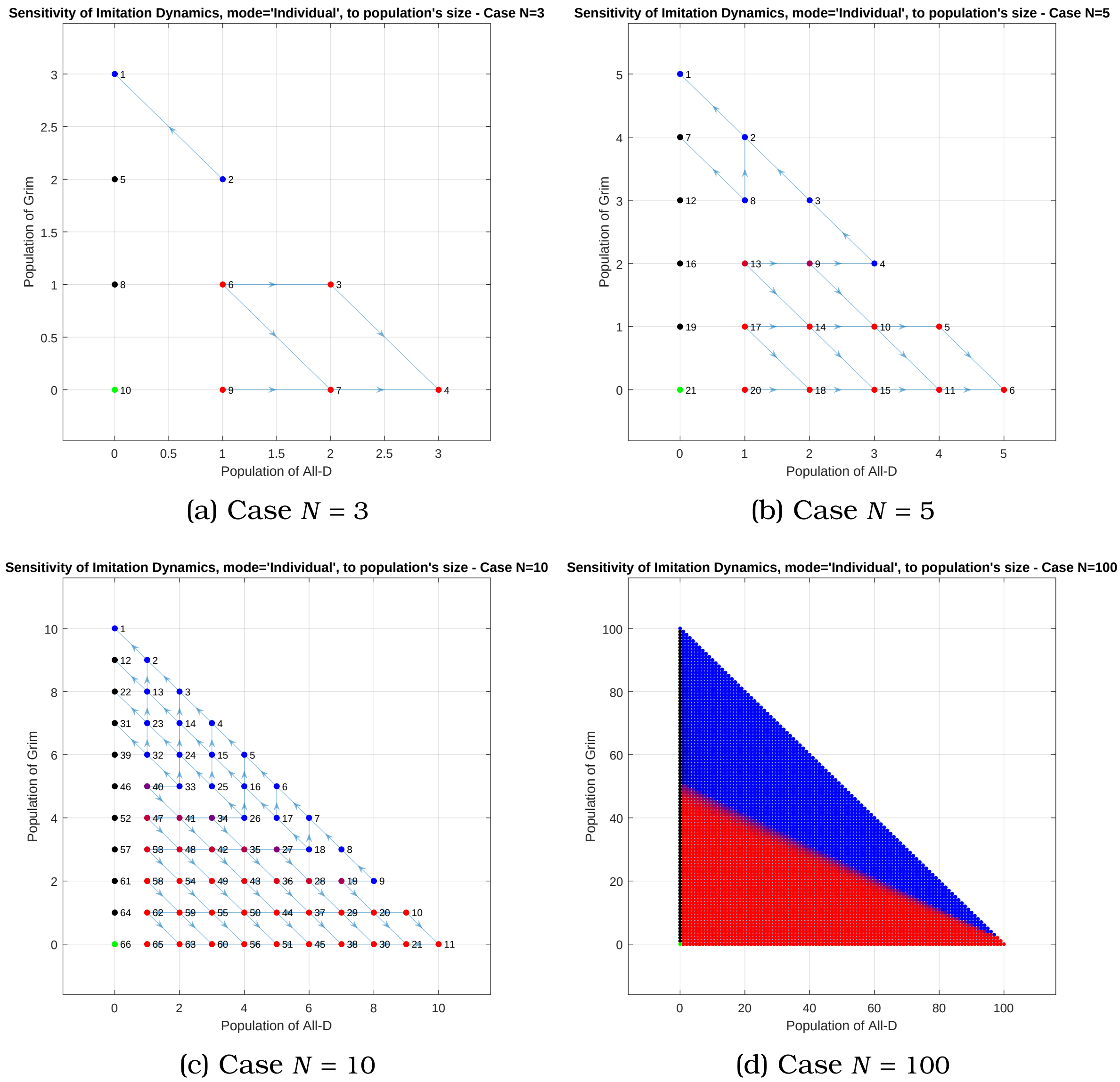


Figure 21: Testing “Total” mode for best strategy calculation with $POP0 = \begin{bmatrix} 1 & 4 & 5 \end{bmatrix}$



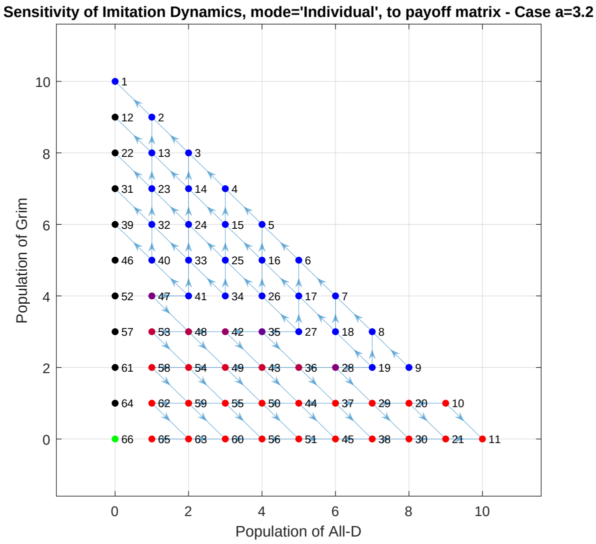
(a) Case $N = 3$

(b) Case $N = 5$

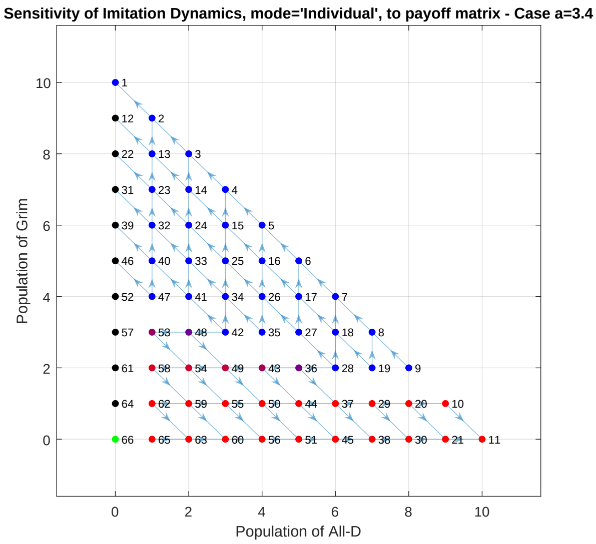
(c) Case $N = 10$

(d) Case $N = 100$

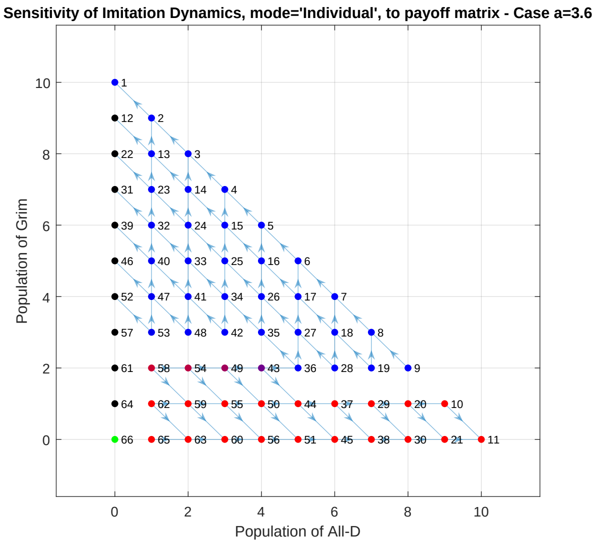
Figure 22: Sensitivity of Imitation Dynamics (mode="Individual") to population size for $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$



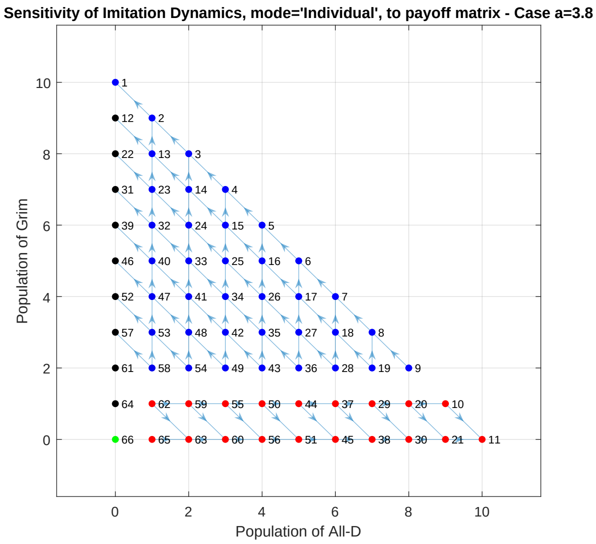
(a) Case $a = 3.2$



(b) Case $a = 3.4$

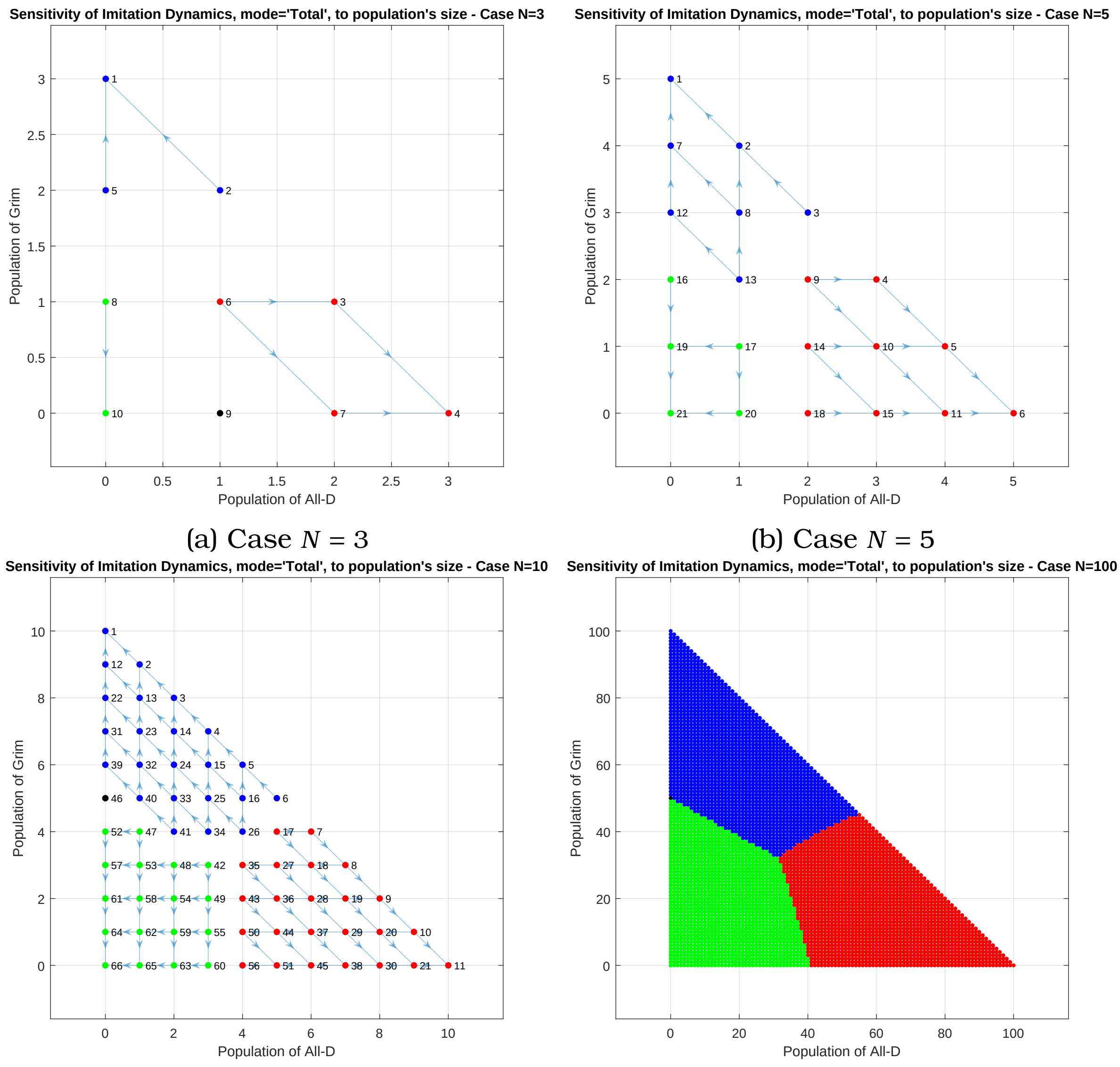


(c) Case $a = 3.6$



(d) Case $a = 3.8$

Figure 23: Sensitivity of Imitation Dynamics (mode="Individual") to payoff matrix parameter a for $B = \begin{bmatrix} a & 1 \\ 4 & 2 \end{bmatrix}$ and $N = 10$



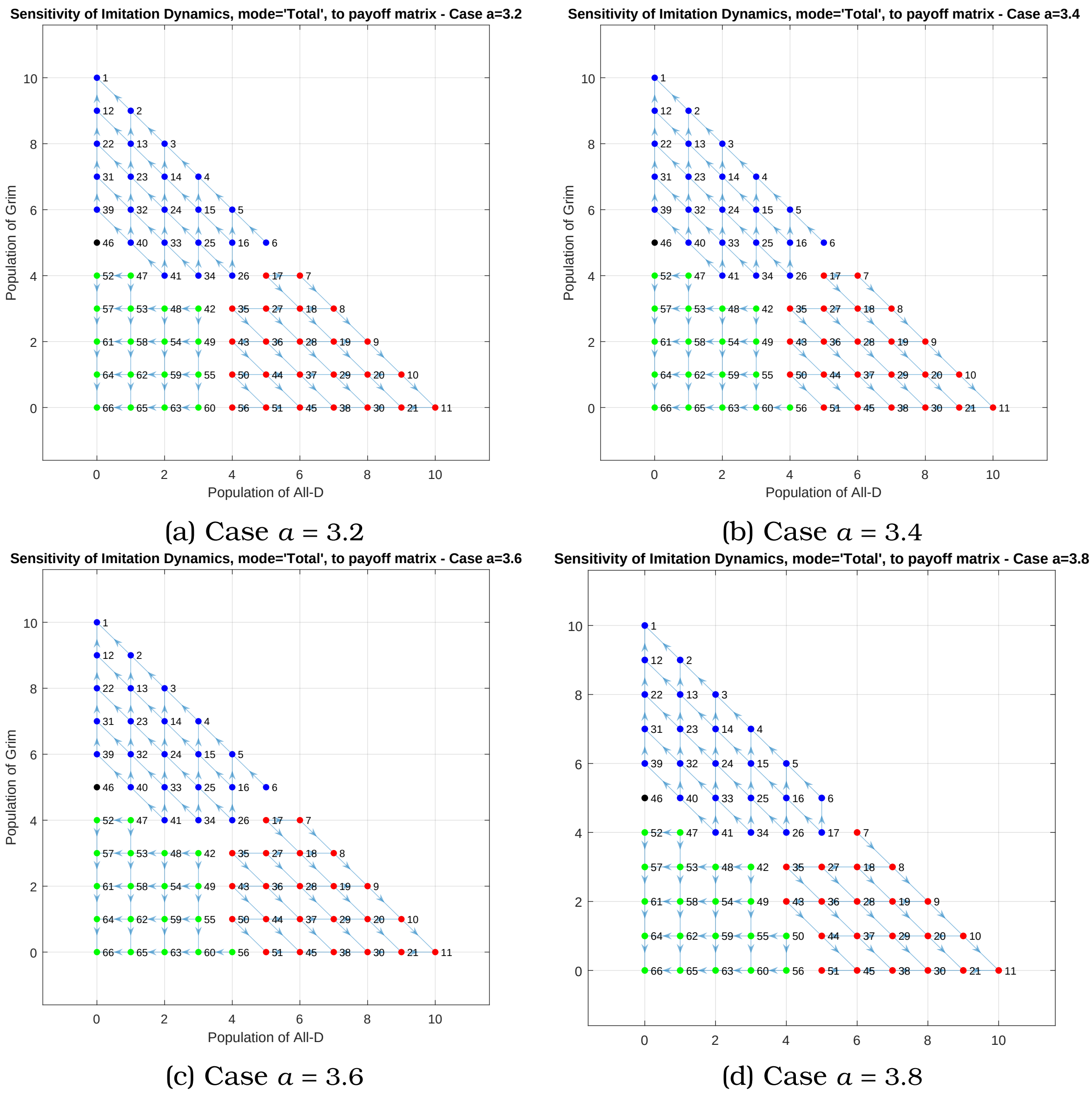
(a) Case $N = 3$

(b) Case $N = 5$

(c) Case $N = 10$

(d) Case $N = 100$

Figure 24: Sensitivity of Imitation Dynamics (mode="Total") to population size for $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$



(a) Case $a = 3.2$

(b) Case $a = 3.4$

(c) Case $a = 3.6$

(d) Case $a = 3.8$

Figure 25: Sensitivity of Imitation Dynamics (mode="Total") to payoff matrix parameter a for $B = \begin{bmatrix} a & 1 \\ 4 & 2 \end{bmatrix}$ and $N = 10$

5 Discussion

5.1 Comparing Fitness vs Imitation Dynamics

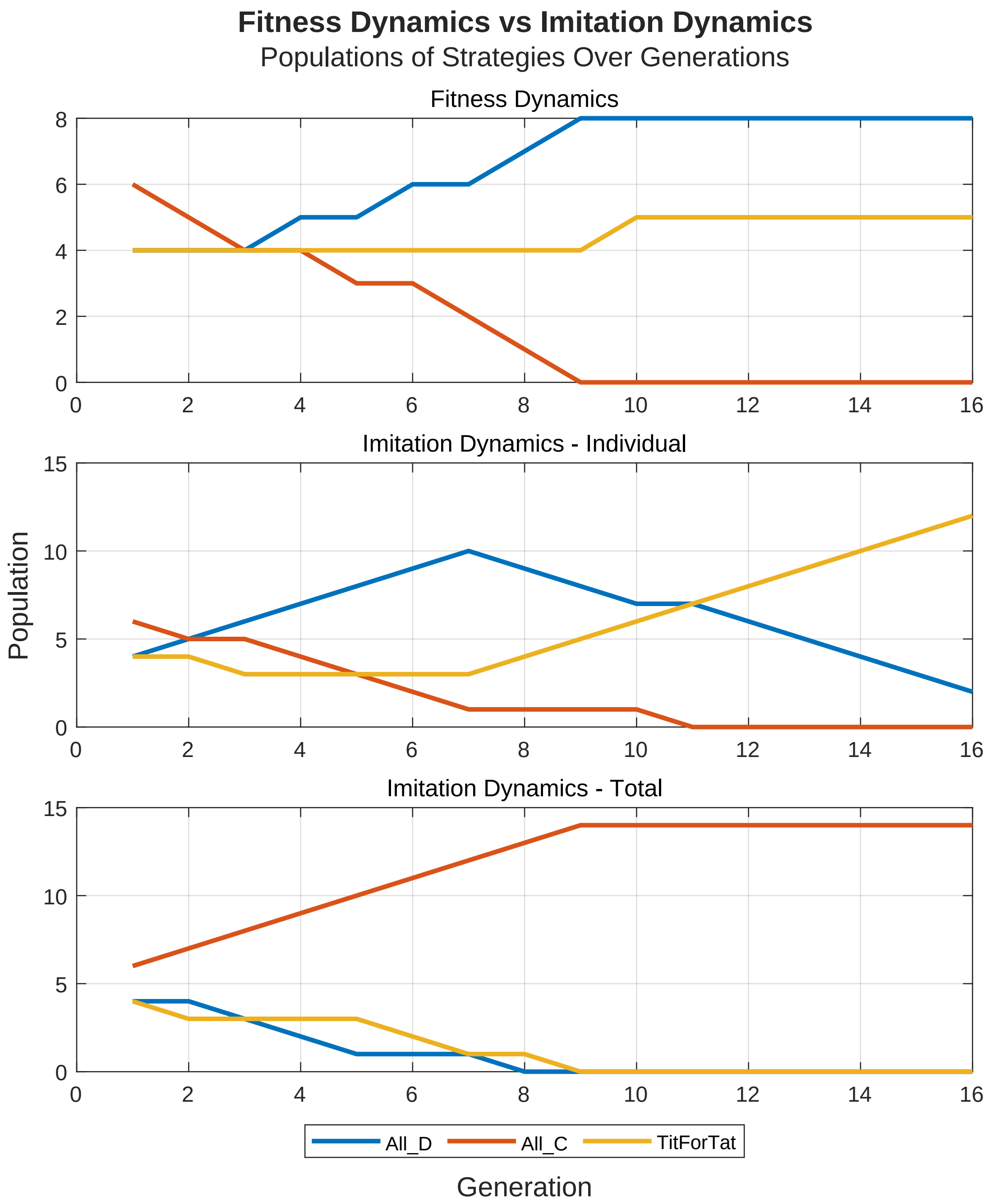


Figure 26: Fitness Dynamics vs Imitation Dynamics $\begin{bmatrix} All\_D & All\_C & TitForTat \end{bmatrix}$ $POP0 = \begin{bmatrix} 4 & 6 & 4 \end{bmatrix}$

In Figure 26, a comparison is made between the results of TourSimFit, TourSimImi with mode = 'Individual', and TourSimImi with mode = 'Total' for the strategies

$$\begin{bmatrix} \text{All\_D} & \text{All\_C} & \text{TitForTat} \end{bmatrix}$$

with initial population

$$POP0 = \begin{bmatrix} 4 & 6 & 4 \end{bmatrix}.$$

We observe that the Fitness case resembles the Individual mode of the imitation dynamics. However, a key difference is noted: in the imitation dynamics, only one strategy survives. The Total mode exhibits a similar evolution to the Individual mode, but we observe that a different strategy survives—specifically, All_C. This occurs because it has a higher initial population compared to the other strategies, and thus is better able to take advantage of the cooperative strategy TitForTat through the mechanism of the Total mode in identifying the best-performing strategy. This is obviously just an examplary simulation and there are way more comparisons to be made, which are left as a fun puzzle for the reader. Run example37 of the Examples folder (after reading Quickstart guide) to recreate the figure.

5.2 Conclusions

From the analysis of the Evolutionary Games we conclude the following:

1. The two dynamics offer different outlooks to the evolutionary aspect of the game, thus creating vastly different results.

2. Even for the same dynamics, minor changes like the ones in "compensation" mode or "Individual" vs "Total" the results can change drastically.

3. Important aspects to be considered for the results are the strategies used, the initial population, the payoff matrix, the game length and the population magnitude.

Possible improvements of this toolbox include:

1. The inclusion of proper handling for random strategies. Currently it is impossible because of a fast but deterministic way to calculate payoffs for matches between all strategies.

2. More comparisons between the two dynamics and overall more interesting examples.

3. Usage of sparse matrices for faster handling in the state transition matrix and faster execution of the functions.

Feel free to use all or parts of this toolbox, making sure to mention it was created by us.

6 References

[1] J. McKenzie Alexander. Evolutionary Game Theory. Cambridge University Press, 2023.

[2] Robert Axelrod and Douglas Dion. The further evolution of cooperation. Science, 242(4884):1385–1390, 1988.

[3] Robert Axelrod and William D. Hamilton. The evolution of cooperation. Science, 211(4489):1390–1396, 1981.

[4] Philippe Mathieu, Bruno Beaufils, and Jean-Paul Delahaye. Studies on dynamics in the classical iterated prisoner's dilemma with few strategies. In European Conference on Artificial Evolution, Berlin Heidelberg, 1999. Springer.

7 Appendices

7.1 Documentation

7.1.1 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.

7.1.2 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.

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).

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.

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), $POP1$ the population matrix for each generation of the Fitness Dynamics (created by TourSimFit), $POP2$ the population matrix for each generation of the Imitation Dynamics - "Individual" mode (created by TourSimImi), $POP3$ 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.

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.

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), $POP1$ the theoretical population matrix for each generation of the Fitness Dynamics (created by TourTheFit), $POP2$ the population matrix for each generation of the Fitness Dynamics without compensation (created by TourSimFit), $POP3$ 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.

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.

PlotStateTransitionGraph

The PlotStateTransitionGraph function is responsible for visualizing the directed transition graph of the Markov Chain that governs the Imitation Dynamics. It plots each state as a node in the state space and colors them based on their long-term transition probabilities to key absorbing states (specifically the homogeneous populations of each pure strategy).

It requires the following inputs: P – The transition matrix of the Markov Chain, calculate by TourTheImi. $POP0$ – The $1 \times S$ initial population vector, where S is the number of strategies. The entries specify the count of agents using each strategy. *Strategies* – A cell array of strings containing the names of each strategy used in the simulation (e.g., 'All_D', 'All_C', 'TFT'). *Title* – A string used as the title of the plot and as the name of the exported PDF file.

Internally, the function performs the following steps. It generates the entire state space using the number of strategies and the total population size, calculates the probability distribution over time by exponentiating the transition matrix to a high power (P^{100}), effectively estimating the steady-state transition probabilities. Then isolates the probabilities of each state transitioning to the three key homogeneous states (e.g., all agents being All_D, All_C, or TFT) and normalizes these probabilities and constructs a directed graph where each node represents a state and each directed edge represents a possible transition. Self-loops are removed for clarity. Node colors are mapped to the normalized probability distribution over the selected absorbing states. The graph is plotted and exported as a vector PDF to the location specified by the global variable figurepath.

The function returns a matrix SC, where each row corresponds to a state and each column represents the normalized probability of transitioning to one of the three homogeneous strategy states. This allows further numerical analysis or custom visualizations if needed.

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.

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).

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), $POP0$ 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 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.

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), $POP0$ 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 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.

7.1.3 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.

example01

The first example shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in Figure 1, with Title “Defectors may be strong”. The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$, $Strategies = ["per\_ddc", "Alternator", "soft\_majo"]$, initial population $POP0 = [100; 100; 100]$, number of rounds per match $T = 1000$ and number of generations $J = 90$.

example02

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in Figure 2, with Title “Monotonous convergence”. The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$, $Strategies = ["Grim", "TitForTat", "Alternator"]$, initial population $POP0 = [100; 100; 100]$, number of rounds per match $T = 1000$ and number of generations $J = 10$.

example03

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in Figure 3, with Title “Attenuated oscillatory movements”. The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$, $Strategies = ["per\_ccd", "per\_ddc", "soft\_majo"]$, initial population $POP0 = [450; 1000; 100]$, number of rounds per match $T = 1000$ and number of generations $J = 420$.

example04

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in Figure 4, with Title “Periodic movements”. The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$, $Strategies = ["per\_ccd", "per\_ddc", "soft\_majo"]$, initial population $POP0 = [300; 200; 100]$, number of rounds per match $T = 1000$ and number of generations $J = 1000$.

example05

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in Figure 5, with Title “Increasing oscillations”. The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$, $Strategies = ["Alternator", "per\_ddc", "soft\_majo"]$, initial population $POP0 = [400; 300; 200]$, number of rounds per match $T = 1000$ and number of generations $J = 450$.

example06

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in Figure 6, with Title “Disordered oscillations”. The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$, $Strategies = [\text{"}soft\_majo\text{"}, \text{"}per\_ccccd\text{"}, \text{"}Prober\text{"}]$, initial population $POP0 = [100; 500; 800]$, number of rounds per match $T = 1000$ and number of generations $J = 280$.

example07

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in the left plot of Figure 7, with Title “Sensitivity of dynamics to population’s size”. The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$, $Strategies = [\text{"}per\_ccd\text{"}, \text{"}soft\_majo\text{"}, \text{"}per\_ddc\text{"}]$, initial population $POP0 = [300; 100; 244]$, number of rounds per match $T = 1000$ and number of generations $J = 1000$.

example08

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in the right plot of Figure 7, with Title “Sensitivity of dynamics to population’s size”. The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$, $Strategies = [\text{"}per\_ccd\text{"}, \text{"}soft\_majo\text{"}, \text{"}per\_ddc\text{"}]$, initial population $POP0 = [300; 100; 245]$, number of rounds per match $T = 1000$ and number of generations $J = 1000$.

example09

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in the left plot of Figure 8, with Title “Sensitivity of winner to population’s size”. The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$, $Strategies = [\text{"}per\_ddc\text{"}, \text{"}soft\_majo\text{"}, \text{"}Alternator\text{"}]$, initial population $POP0 = [100; 159; 100]$, number of rounds per match $T = 1000$ and number of generations $J = 120$.

example10

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in the right plot of Figure 8, with Title “Sensitivity of winner to population’s size". The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$, $Strategies$ = [*"per_ddc", "soft_majo", "Alternator"*], initial population $POP0 = [100; 160; 100]$, number of rounds per match $T = 1000$ and number of generations $J = 120$.

example11

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in the left plot of Figure 9, with Title "Sensitivity to game length". The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$, $Strategies$ = [*"per_ccd", "soft_majo", "per_ddc"*], initial population $POP0 = [300; 100; 244]$, number of rounds per match $T = 7$ and number of generations $J = 1000$.

example12

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in the right plot of Figure 9, with Title "Sensitivity to game length". The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$, $Strategies$ = [*"per_ccd", "soft_majo", "per_ddc"*], initial population $POP0 = [300; 100; 244]$, number of rounds per match $T = 6$ and number of generations $J = 1000$.

example13

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in the left plot of Figure 10, with Title "Sensitivity to CIPD payoff". The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 4.6 & 1 \end{bmatrix}$, $Strategies$ = [*"per_ccd", "soft_majo", "per_ddc"*], initial population $POP0 = [300; 100; 244]$, number of rounds per match $T = 1000$ and number of generations $J = 475$.

example14

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in the right plot of Figure 10, with Title "Sensitivity to CIPD payoff". The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 4.7 & 1 \end{bmatrix}$, $Strategies$ = [*"per_ccd", "soft_majo", "per_ddc"*], initial population $POP0 = [300; 100; 244]$, number of rounds per match $T = 1000$ and number of generations $J = 1000$.

example15

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in the plots of Figure 11, with Title “Sensitivity to repartition computation method”. The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$, $Strategies$ = [”per_ccd”,-”$soft\_majo$”, ”per_ddc”], initial population $POP0 = [300; 100; 200]$, number of rounds per match $T = 1000$ and number of generations $J = 1000$. Note that the plot on the left corresponds to the plot created by TourSimFit and the plot on the right corresponds to the one created by TourTheFit.

example16

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in the left plot of Figure 12, with Title “Sensitivity to repartition computation method”. The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 4.6 & 1 \end{bmatrix}$, $Strategies$ = [”per_ccd”, ”$soft\_majo$”, ”per_ddc”], initial population $POP0 = [450; 100; 1000]$, number of rounds per match $T = 1000$ and number of generations $J = 1000$.

example17

This shows the results of TourTheFit, TourSimFit with compensation and TourSimFit without compensation in the case presented by Mathieu et al in the right plot of Figure 12, with Title “Sensitivity to repartition computation method”. The functions are called with payoff matrix $B = \begin{bmatrix} 3 & 0 \\ 4.6 & 1 \end{bmatrix}$, $Strategies$ = [”per_ccd”, ”$soft\_majo$”, ”per_ddc”], initial population $POP0 = [45; 10; 100]$, number of rounds per match $T = 1000$ and number of generations $J = 1000$.

example18

This showcases the sensitivity of Imitation Dynamics, mode = “Individual” to the population’s size, in the case $N = 3$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies$ = [”All_C”, ”All_D”, ”$Grim$”], initial population $POP0 = [0; 0; 3]$, number of imitators per round $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0; 0; 3]$, $Strategies$ = [”All_C”, ”All_D”, ”$Grim$”] and a proper title.

example19

This showcases the sensitivity of Imitation Dynamics, mode = “Individual” to the population’s size, in the case $N = 5$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = ["All\_C", "All\_D", "Grim"]$, initial population $POP0 = [0; 0; 5]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0; 0; 5]$, $Strategies = ["All\_C", "All\_D", "Grim"]$ and a proper title.

example20

This showcases the sensitivity of Imitation Dynamics, mode = “Individual” to the population’s size, in the case $N = 10$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = ["All\_C", "All\_D", "Grim"]$, initial population $POP0 = [0; 0; 10]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0; 0; 10]$, $Strategies = ["All\_C", "All\_D", "Grim"]$ and a proper title.

example21

This showcases the sensitivity of Imitation Dynamics, mode = “Individual” to the population’s size, in the case $N = 100$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = ["All\_C", "All\_D", "Grim"]$, initial population $POP0 = [0; 0; 100]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0; 0; 100]$, $Strategies = ["All\_C", "All\_D", "Grim"]$ and a proper title.

example22

This showcases the sensitivity of Imitation Dynamics, mode = “Individual” to the payoff matrix, in the case $a = 3.2$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3.2 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = ["All\_C", "All\_D", "Grim"]$, initial population $POP0 = [0; 0; 10]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0; 0; 10]$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$ and a proper title.

example23

This showcases the sensitivity of Imitation Dynamics, mode = “Individual” to the payoff matrix, in the case $a = 3.4$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3.4 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$, initial population $POP0 = [0; 0; 10]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0; 0; 10]$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$ and a proper title.

example24

This showcases the sensitivity of Imitation Dynamics, mode = “Individual” to the payoff matrix, in the case $a = 3.6$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3.6 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$, initial population $POP0 = [0; 0; 10]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0; 0; 10]$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$ and a proper title.

example25

This showcases the sensitivity of Imitation Dynamics, mode = “Individual” to the payoff matrix, in the case $a = 3.8$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3.8 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$, initial population $POP0 = [0; 0; 10]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0; 0; 10]$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$ and a proper title.

example26

This showcases the sensitivity of Imitation Dynamics, mode = “Total” to the population’s size, in the case $N = 3$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$, initial population $POP0 = [0;0;3]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0;0;3]$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$ and a proper title.

example27

This showcases the sensitivity of Imitation Dynamics, mode = "Total" to the population's size, in the case $N = 5$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$, initial population $POP0 = [0;0;5]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0;0;5]$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$ and a proper title.

example28

This showcases the sensitivity of Imitation Dynamics, mode = "Total" to the population's size, in the case $N = 10$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$, initial population $POP0 = [0;0;10]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0;0;10]$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$ and a proper title.

example29

This showcases the sensitivity of Imitation Dynamics, mode = "Total" to the population's size, in the case $N = 100$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$, initial population $POP0 = [0;0;100]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0;0;100]$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$ and a proper title.

example30

This showcases the sensitivity of Imitation Dynamics, mode = “Total” to the payoff matrix, in the case $a = 3.2$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3.2 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$, initial population $POP0 = [0;0;10]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0;0;10]$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$ and a proper title.

example31

This showcases the sensitivity of Imitation Dynamics, mode = “Total” to the payoff matrix, in the case $a = 3.4$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3.4 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$, initial population $POP0 = [0;0;10]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0;0;10]$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$ and a proper title.

example32

This showcases the sensitivity of Imitation Dynamics, mode = “Total” to the payoff matrix, in the case $a = 3.6$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3.6 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$, initial population $POP0 = [0;0;10]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0;0;10]$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$ and a proper title.

example33

This showcases the sensitivity of Imitation Dynamics, mode = “Total” to the payoff matrix, in the case $a = 3.8$. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3.8 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [\text{"}All\_C\text{"}, \text{"}All\_D\text{"}, \text{"}Grim\text{"}]$, initial population $POP0 = [0;0;10]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function PlotStateTransitionGraph is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [0;0;10]$, $Strategies = [''All\_C'', ''All\_D'', ''Grim'']$ and a proper title.

example34

This is an examplary showcase of TourTheImi, AnalyzeMarkovChain and TourSimImi. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [''All\_D'', ''All\_C'', ''TitForTat'']$, initial population $POP0 = [1;5;3]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function AnalyzeMarkovChain is then called with arguments state transition matrix $P1$ as calculated by TourTheImi, initial population $POP0 = [1;5;3]$, $Strategies = [''All\_D'', ''All\_C'', ''TitForTat'']$ and a proper title. Lastly, TourSimImi is called with the same arguments as TourTheImi and the populations are plotted.

example35

This is an examplary showcase of the default ("Individual") mode for best strategy calculation. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [''All\_D'', ''All\_C'', ''TitForTat'']$, initial population $POP0 = [1;4;5]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function AnalyzeMarkovChain is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [1;5;3]$, $Strategies = [''All\_D'', ''All\_C'', ''TitForTat'']$ and a proper title.

example36

This is an examplary showcase of the secondary ("Total") mode for best strategy calculation. The function TourTheImi is called with payoff matrix $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies = [''All\_D'', ''All\_C'', ''TitForTat'']$, initial population $POP0 = [1;4;5]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$, number of generations $J = 15$ and an extra argument $mode = ``Total''$. The function AnalyzeMarkovChain is then called with arguments state transition matrix P as calculated by TourTheImi, initial population $POP0 = [1;5;3]$, $Strategies = [''All\_D'', ''All\_C'', ''TitForTat'']$ and a proper title.

example37

This is an examplary comparison between Fitness and Imitation Dynamics. The function TourSimFit is called with payoff matrix $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies$ = [$''All\_D''$, $''All\_C''$, $''TitForTat''$], initial population $POP0 = [4; 6; 4]$, number of rounds per match $T = 100$ and number of generations $J = 15$. The function TourSimImi is called twice with arguments $B = \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix}$, $Strategies$ = [$''All\_D''$, $''All\_C''$, $''TitForTat''$], initial population $POP0 = [1; 4; 5]$, number of imitators per generation $K = 1$, number of rounds per match $T = 100$ and number of generations $J = 15$, once with an extra argument $mode$ = "$Total''$ and once without it (to simulate both with "Individual" mode as well as "Total" mode). The results of each simulation are then plotted accordingly.

7.1.4 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). For the strategies below, C stands for Cooperation and D stands for Defection.

All_C

Strategy that always cooperates.

All_D

Strategy that always defects.

Alternator

Strategy that alternates between cooperating and defecting, starting with cooperation in the first round.

Cycler

Strategy that permanently cycles the moves C, C, C, D.

Detective

Strategy that starts with the moves C, D, C, C. If the opponent defects at least once in these first four rounds, play as TitForTat forever. Else, always defect.

Grim

Strategy that always cooperates unless the opponent defects: after the first opponent defection, always defect.

per_ccccd

Strategy that permanently cycles the moves C, C, C, C, D.

per_ccd

Strategy that permanently cycles the moves C, C, D.

per_ddc

Strategy that permanently cycles the moves D, D, C.

Prober

Strategy that starts with the moves D, C, C. If the opponent cooperates at moves 2 and 3, plays defect for the rest of the game. Otherwise plays as TitForTat.

SneakyTitForTat

Strategy that starts by cooperating in the first round and playing as TitForTat in the second round. If the opponent has not defected in these first two rounds, defect once and check the opponent's response. If the opponents retaliates, play a cooperation to repent and then play as TitForTat. If the opponent does not retaliate, keep playing defect and checking the opponent's response as before.

soft_majo

Strategy that plays the same move as the majority of the opponent's moves up until the previous round. In case of equality, cooperate.

SpitefulTitForTat

Strategy that starts by cooperating and then plays like TitForTat, until the opponent defects twice in a row, in which case the player always defects.

TitForTat

Strategy that cooperates in the first move and for the rest of the game plays the opponent's previous move.

TitForTwoTats

Strategy that plays defect only if the opponent has defected twice in a row. Otherwise, always cooperates.

TwoTitsForTat

Strategy that starts by cooperating and replies to each opponent defection by two defections of its own. More specifically, if there has been a defection by the opponent in the two previous rounds, the player chooses defect. Otherwise, plays cooperate.

7.2 GitHub repository

For the full source code visit the GitHub repository:
https://github.com/mdelopo/EvolutionaryGamesToolbox

List of Figures