Worked example (Example 1)

Nadiah Kristensen

December 11, 2025

This document refers to the contents of `examples/deterministic/example_1/`, which contains a worked example of solving the introspection dynamics for a 2-player game.

Contents

1 Scenario used for this example

The scenario modelled is a two-player game with parameter values chosen such: the players are symmetric, it is always in the players' self-interest to first-author, it is not in the players' self-interest to coauthor. The payoff structure for the coauthorship decision is a Prisoner's Dilemma.

For the introspection dynamics, I assume that both players will never first-author against their self-interest (first-author strategy "`never`"), and therefore only the dynamics of co-authorship strategies are considered. The space of possible co-authorship strategies considered is: `pavlov_c`, `pavlov_d`, `all_c`, and `all_d`.

2 Identify permitted action-state transitions and count implementation errors

As described in `docs/deterministic/used_to_build.pdf`, when considering within-game transitions between action states with implementation errors, certain transitions between the deterministic and ultimate action-states $A_d \rightarrow A_u$ are permitted and others are not. For permitted transitions, the number of errors required to perform the transition is the input into the calculation for the probability of that transition occurring, which in turn is used to calculate the within-game stationary distribution.

Which transitions are permitted and the number of errors required for the transition is independent of the parameters of the model apart from the number of players n. Therefore, a script was written to pre-identify permitted transitions and calculate their number of implementation errors.

The script `scripts/deterministic/calc_transitions_errors.py` produces a `.parquet` file for the two-player game, `results/deterministic/transitions_errors_nbr_players_2.parquet`. The first few rows of the file are shown below:

```
from_state, to_state, is_valid, nbr_errors, nbr_correct
         0,        0,     True,          0,           2
         1,        0,     True,          1,           1
         2,        0,    False,          0,           2
         3,        0,    False,          1,           1
         4,        0,    False,          0,           2
         5,        0,    False,          1,           1
         6,        0,    False,          0,           2
...
```

3 Calculate payoffs to each player in each action state

The script `examples/deterministic/example_1/calc_actions_payoffs.py` was used to calculate the payoffs to each player in each possible action-pair state.

Key steps include specifying the game parameters:

```
# game parameters
n = 2  # number of players
b = 2  # maximum benefit of a paper
c_f = 0.9  # cost of first-authoring
c_c = 0.8  # cost of co-authoring

# topic preferences
preferences = np.array([[0], [2]])
assert len(preferences) == n
```

Getting every 2-player action state (makes use of `functions/model_fncs.py` imported as `mf`):

```
ID_2_actions = mf.get_ID_2_actions(n)
```

Calculating payoffs:

```
topics = mf.topics_fnc(params)
paysV = [mf.calc_pays(params, ID_2_actions[ID], topics) for ID in IDs]
```

4 Caclulate within-game deterministic transitions between action states

Across the rounds of the iterated game, the strategy of each player determines their action in the next round, which is based on the actions taken in the previous round. The previous-round actions considered include both their own actions as well as those of the the other players. Therefore, each possible strategy pairing results in a deterministic dynamic of transitions between different action states.

The script `examples/deterministic/example_1/calc_within_game_deterministic_transitions.py` calculates the deterministic transitions between action states for every pairing of coauthorship strategies. For each coauthor-strategy pair, the calculation is a two-step process, as described in `docs/deterministic/used_to_build.pdf`.

First, decisions are made based on self-interest. The self-interested actions are those with a positive action gain (calculated assuming we are moving from non-authorship $a_{i,j} = 0$ to authorship $a_{i,j} = 1$). In `calc_within_game_deterministic_transitions.py`, the positive action gains are identified with

```
action_gains_positive = mf.calc_action_gains_positive(params, ID_2_actions,
   gammas, alignments)
```

`action_gains_positive` is an input into the function `calc_deterministic_transitions()`, which calculates the deterministic transitions matrix

```
deterministic_transitions_matrix = mf.calc_deterministic_transitions(
    params, actions_2_ID, action_gains_positive, fsauthor_strats,
        coauthor_strats
)
```

The function `calc_deterministic_transitions()` is found in `functions/model_fncs.py`, where the two steps can be seen. For both first-author and coauthor decisions, first one determines if the action is in the player's self-interest. If it is not, then the first-author or coauthor strategy is used to determine the next action.

The script `examples/deterministic/example_1/plot_all_within_game_deterministic_transitions.py` prints graphs of the action transitions for coauthor-strategy pairs. All graphs can be found in PDF files named `examples/deterministic/example_1/within_game_deterministic_transitions_{strategy_1}_Vs_{strategy_2}.pdf`, and two selected examples are shown in Fig. 1.

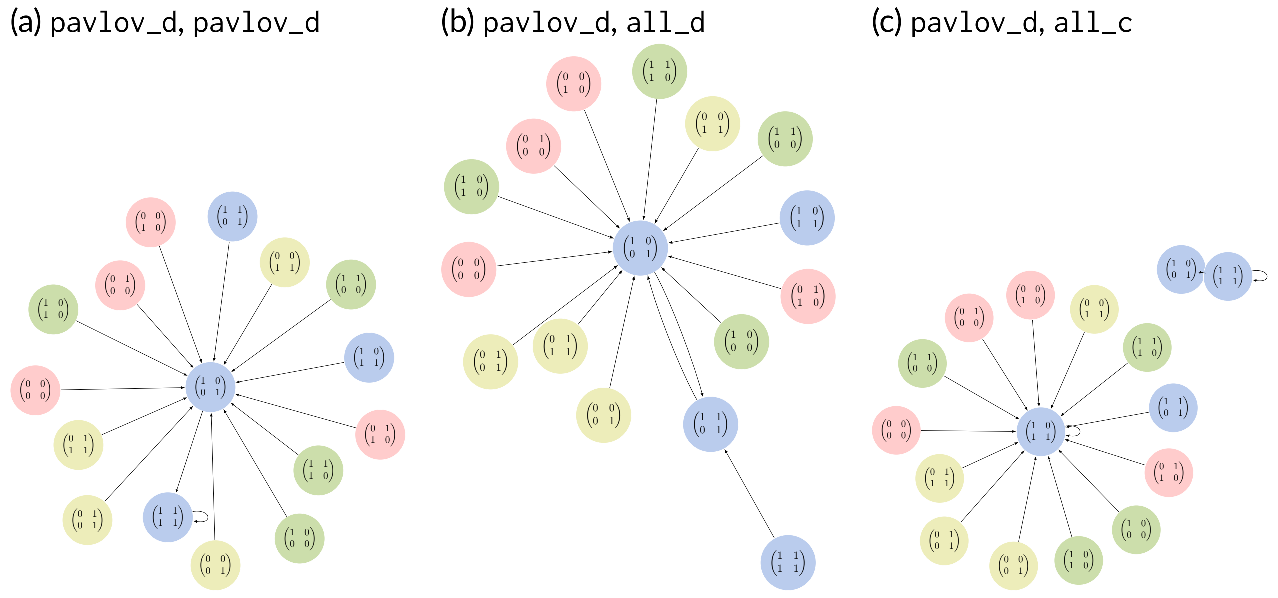

Figure 1: Deterministic transitions between action states selected pairs of strategies. Nodes represent an action-state A, which in this 2-player game is a 2×2 binary matrix, and directed edges indicate how the actions of all players change at each round of the game.

When a player pursuing a Pavlov coauthorship strategy is paired with another Pavlov player, they will settle into a state of mutually coauthorship (Fig. 1a), whereas a pairing with a player who never coauthors results in the Pavlov player switching between coauthoring and defecting (Fig. 1b). Some within-game dynamics have more than one attractor, e.g., the pairing of a Pavlov player with a player who always coauthors (Fig. 1c). In situations with more than one attractor, a stationary distribution cannot be found without an additional assumption, such as by introducing a small implementation-error (below).

5 Calculate within-game stationary distribution of action states (analytic solution)

The script examples/deterministic/example_1/calc_within_game_stationary_distributions.py calculates the stationary distribution of action states using the method described this blog post. For each strategy pairing, first, the transition matrix with implementation-error terms ε is generated

```
# get the transition matrix with error terms included (symbolic entries a
   fnc of eps)
P, eps = mf.symbolic_transitions_with_errors(pre_2_det, fname_error_count)
```

Then the analytic stationary distribution as $\varepsilon \to 0$ is calculated

```
# find the stationary distribution as eps -> 0
stationary_distn_sp, pwr_level = mf.find_stationary_distribution(P, eps)
```

The function symbolic_transitions_with_errors() functions/model_fncs.py uses the symbolic mathematics toolbox SymPy to construct the transition matrix with ε terms. The key input needed is the transition_errors_file_name, which in this example is the file results/deterministic/transitions_errors_nbr_players_2.parquet described in Sect. 2.

The function find_stationary_distribution() takes the symbolic matrix P and the symbolic variable eps to construct the simultaneous equations and solve them. The details of the solution are found in the blog post.

The stationary distributions are stored in the parquet file examples/deterministic/example_1/within_game_stationary_distribution.parquet. The first few rows of the 3rd to 6th columns of the file are shown below:

```
pavlov_c pavlov_c [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1] [1 1 1 1 1 1 1 1 1 1 1 1
    1 1 1 1]
pavlov_c pavlov_d [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1] [1 1 1 1 1 1 1 1 1 1 1 1
    1 1 1 1]
pavlov_c all_c    [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 2] [1 1 1 1 1 1 1 1 1 1 1 3
    1 1 1 3]
```

The first two columns shown are the Player 1 and Player 2 coauthorship strategies, respectively, and the last two columns are lists of the the numerators and denominators of the stationary probabilities. List indices correspond to action-states indices. The analytic stationary distribution always returns rational probabilities, so the stationary probabilities can be stored without loss of accuracy.

Note that the script calc_within_game_stationary_distributions.py uses this method for every strategy-pair scenario, including those with a deterministic dynamics with only one attractor. It is likely that it would be more computationally efficient to identify single-attractor cases and use another method to find their attractors.

6 Calculate longterm expected payoffs

For each strategy pairing, the expected payoff to each player is calculated by summing the products of their payoff in each action state by their probability of being in that action state. The script `examples/deterministic/example_1/calc_within_game_stationary_payoffs.py` performs this calculation.

7 Calculate introspection dynamics stationary distribution

To determine how players' strategies evolve over time, an introspection-dynamics model is used. The introspection dynamics are also modelled as Markov chain, so the complete model layers a Markov chain describing the transitions between players' strategies on top of the Markov chain describing the within-game transitions between action states.

Denote the longterm expected payoff to player i when i plays strategy x and j plays strategy y as $\pi_i(\{s_{i,x}, s_{j,y}\})$. Only one player can update their strategy at a time; therefore, only transitions $\{s_{i,x}, s_{j,y}\} \rightarrow \{s_{i,z}, s_{j,y}\}$ are permitted. Then the probability that player i switches their strategy from x to z ($x \neq z$) is

$$\mathbb{P}[\{s_{i,x}, s_{j,y}\} \rightarrow \{s_{i,z}, s_{j,y}\}] = \left(\frac{1}{2}\right)\left(\frac{1}{m_i - 1}\right)\rho\left(\{s_{i,x}, s_{j,y}\}, \{s_{i,z}, s_{j,y}\}\right) \tag{1}$$

where $1/2$ is the probability that player i is chosen to update their strategy, $m_i - 1$ is the number of alternative stragies from among which i chose z as the comparison strategy, and $\rho()$ is the probability i switches from x to z given that i and z were chosen.

$\rho()$ is a function of the difference in long-term expected payoffs between the two strategies

$$\rho\left(\{s_{i,x}, s_{j,y}\}, \{s_{i,z}, s_{j,y}\}\right) = \frac{1}{1 + \exp(-\delta\left[\pi_i(\{s_{i,z}, s_{j,y}\}) - \pi_i(\{s_{i,x}, s_{j,y}\})\right])}$$

where δ is the introspection strength.

The transition matrix describing the introspection dynamics is populated with $\mathbb{P}[\{s_{i,x}, s_{j,y}\} \rightarrow \{s_{i,z}, s_{j,y}\}]$.

7.1 As introspection strength approaches infinity

The script `scripts/deterministic/plot_limit_introspection_stationary_distribution.py` plots graphs of the introspection dynamics in the limit $\delta \rightarrow \infty$. As $\delta \rightarrow \infty$, the function $\rho()$ simplifies to

$$\rho\left(\{s_{i,x}, s_{j,y}\}, \{s_{i,z}, s_{j,y}\}\right) = \begin{cases} 0 & \text{if } \pi_i(\{s_{i,z}, s_{j,y}\}) < \pi_i(\{s_{i,x}, s_{j,y}\}) \\ \frac{1}{2} & \text{if } \pi_i(\{s_{i,z}, s_{j,y}\}) = \pi_i(\{s_{i,x}, s_{j,y}\}) \\ 1 & \text{if } \pi_i(\{s_{i,z}, s_{j,y}\}) > \pi_i(\{s_{i,x}, s_{j,y}\}) \end{cases}$$

The dynamics has two attractors (Fig. 2): a strategy pair playing `all_d`, `all_d`; and drift between combinations of `pavlov_c` and `pavlov_d`.

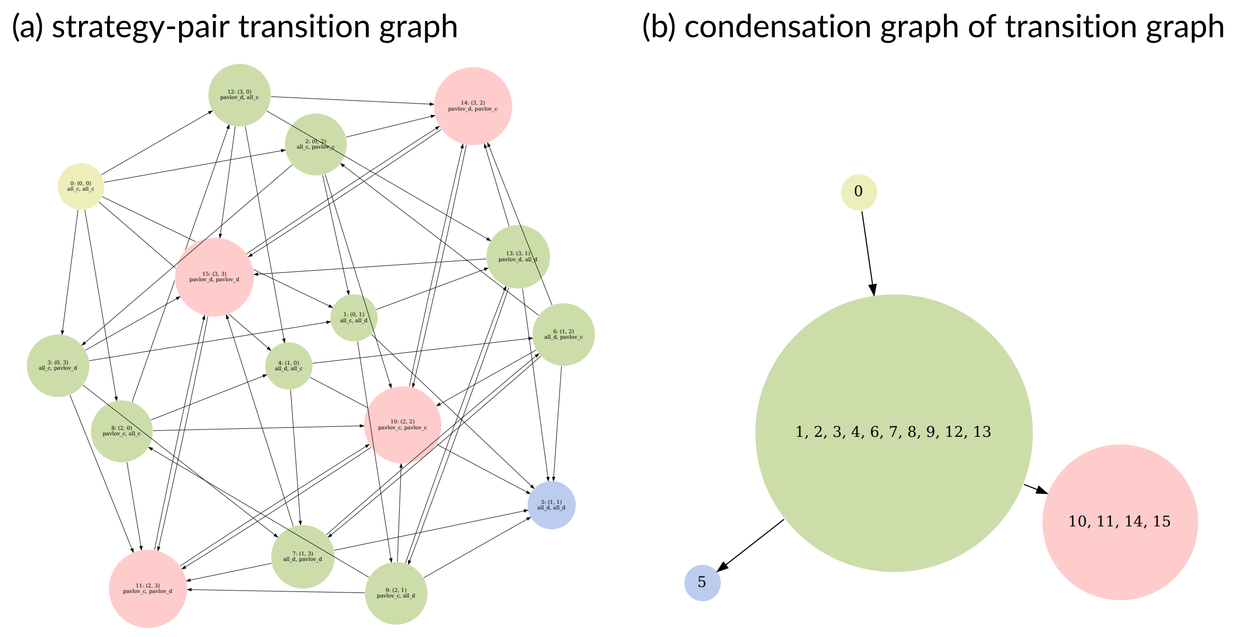
(a) strategy-pair transition graph
(b) condensation graph of transition graph
12: (3, 0) pavlov_d, all_c
14: (3, 2) pavlov_d, pavlov_c
2: (0, 2) all_c, pavlov_c
0: (0, 0) all_c, all_c
13: (3, 1) pavlov_d, all_d
15: (3, 3) pavlov_d, pavlov_d
1: (0, 1) all_c, all_d
6: (1, 2) all_d, pavlov_c
3: (0, 3) all_c, pavlov_d
4: (1, 0) all_d, all_c
10: (2, 2) pavlov_c, pavlov_c
8: (2, 0) pavlov_c, all_c
5: (1, 1) all_d, all_d
7: (1, 3) all_d, pavlov_d
11: (2, 3) pavlov_c, pavlov_d
9: (2, 1) pavlov_c, all_d
0
1, 2, 3, 4, 6, 7, 8, 9, 12, 13
10, 11, 14, 15
5

Figure 2: Graphs summarising the introspection dynamics when introspection strength $\delta \to \infty$. (a) Transitions between strategy pairs. Strongly connected components are indicated by nodes with matching colours. (b) The condensation graph of the transition graph in panel (a).

7.2 For finite introspection strength

The script `scripts/deterministic/calc_numerical_introspection_stationary_distribution.py` calculates the stationary distribution of the introspection dynamics for a range of finite δ values.

The first step is to identify the permissible transitions between strategy-pair states. Then for each permissible transition, the probability that the transition occurs is calculated using Eq. 1

```
mat[old_mat_idx, new_mat_idx] = (1 / 2) * (1 / (nbr_strats - 1)) * \
        prob_transition_given_comparison_pair(delta, focal_idx, old_pays,
           new_pays)
```

where the function `prob_transition_given_comparison_pair()` implements Eq. 7

```
def prob_transition_given_comparison_pair(delta, focal_idx, old_pays,
   new_pays):
    return 1 / (1 + np.exp(-delta * (new_pays[focal_idx] - old_pays[
       focal_idx])))
```

The script `scripts/deterministic/plot_numerical_introspection_stationary_distribution.py` plots the results from `calc_numerical_introspection_stationary_distribution.py` (Fig. 3). Although (`all_d`, `all_d`) is an attractor when $\delta \to \infty$ (Fig. 2), the stationary distribution in the limit appears to be drift between combinations of `pavlov_c` and `pavlov_d` (Fig. 3).

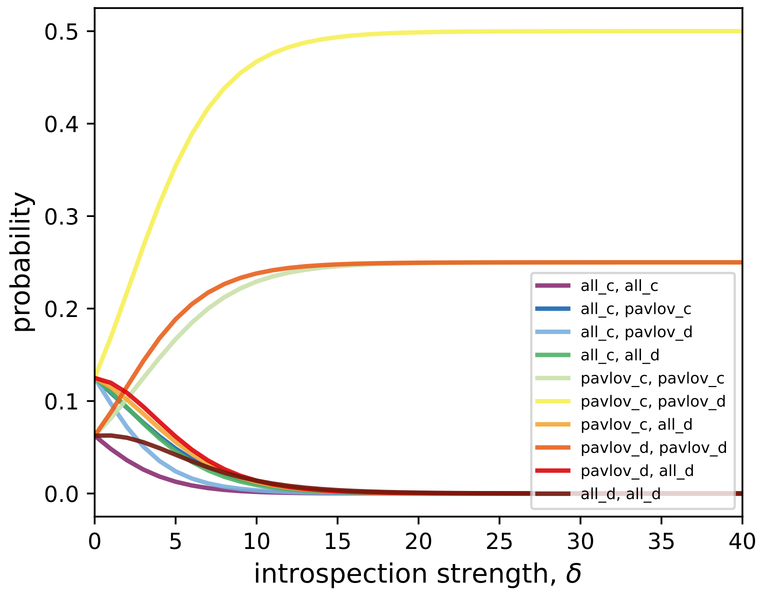

Figure 3: The stationary distribution of strategy-pair probabilities in the introspection dynamics as a function of introspection strength.