BIO445 Quantitative Life Sciences

Game theory - Exercise Sheet

Marco Labarile, marco.labarile@uzh.ch

2022-12-23

Problem 1: Resource competition

Objectives

You will learn how to define and evaluate a strategy in a single 2-player game.

Exercise

1.1

Both you and your partner are each an individual organism that needs to decide every hour (or any other time interval) whether to transform glucose into ATP through respiration or fermentation. Respiration is slow but has a high ATP yield. Fermentation is fast but has a low yield. The problem that you face is that other organisms (your partner) also compete for the same glucose molecules. The following payoff matrix shows one feasible example of your and your partner's ATP gain for different strategies. The numbers symbolize payouts (in energy, food, fitness or some other advantage). The left number is the payout to the player playing on the rows, while the right number is the payout to the player on the columns.

Does this table make sense biologically? Look at the assigned numbers and discuss with your partner.

Table 1: The prisoner's dilemma.

	Respirate (Cooperate)	Ferment (Defect)
Respirate (Cooperate)	3 / 3	0 / 5
Ferment (Defect)	5 / 0	1 / 1

1.2

What is your best strategy given that you cannot influence your partner's choice? First, discuss with your partner. Then we will all briefly discuss. After our group discussion, take 5 minutes to watch https://www.youtube.com/watch?v=t9Lo2fgxWHw.

Problem 2

Objectives

You will learn how to define and evaluate a strategy in an iterated 2-player game.

Exercise

2.1

As in Problem 1, both you and your partner are each an individual organism that needs to decide whether to transform glucose into ATP through respiration or fermentation. However, as in real life, you don't interact only once but multiple times. For example, every hour both of you decide whether to use respiration or fermentation within the next hour. The payoff matrix from Problem 1 still describes your and your partner's ATP gain. Your goal is to maximize your total payoff over a certain number of rounds.

IMPORTANT: It is NOT your goal to beat your partner. Your goal is to end up with the highest payoff for yourself. For example, if both you and your partner always defect each other, you don't lose to each other but other organisms in your neighborhood gain much more ATP and have therefore a higher chance of survival.

You and your partner each write a strategy function that takes as input two vectors called `ownpastactions` and `partnerpastactions`. Each vector contains Boolean variables, which can be either `TRUE` or `FALSE`, that describe the choices you (`ownpastactions`) and your partner (`partnerpastactions`) have made in previous rounds. Let's agree that TRUE stands for respiration (Cooperate) and FALSE stands for fermentation (Defect). Based on these arrays, your strategy function makes a decision on whether you cooperate or defect in the next round. Once both you and your partner wrote your functions, you will play 10 rounds against each other. Here are some sample strategy functions. Try to understand the respective strategies.

```r
strategy1 <- function(ownpastactions, partnerpastactions) {
    return(TRUE)
}

strategy2 <- function(ownpastactions, partnerpastactions) {
    return(runif(1) > 0.75)
}

strategy3 <- function(ownpastactions, partnerpastactions) {
    n <- length(ownpastactions)

    if (n%%3 == 0) {
        return(TRUE)
    } else {
        return(FALSE)
    }
}
```

2.2

The following function evaluates the result of a single round. It takes as input both actions (each TRUE or FALSE) as well as a payout vector, and it returns a vector of length 2 with both your payoffs in this round.

```r
payout <- function(action_p1, action_p2, payoutvector = c(5,
    3, 1, 0)) {
    # best (5) > good (3) > bad (1) > worst (0)
    if (action_p1 & action_p2)
        return(c(payoutvector[2], payoutvector[2]))
    if (action_p1 & !action_p2)
        return(c(payoutvector[4], payoutvector[1]))
    if (!action_p1 & action_p2)
        return(c(payoutvector[1], payoutvector[4]))
    if (!action_p1 & !action_p2)
        return(c(payoutvector[3], payoutvector[3]))
}
```

To simulate N rounds of the game, you will have to write a function `simulate_2P_PD` that takes as input both your strategy functions, a number of rounds `N`, and a payout vector and returns a vector of length 2 with both your total payoffs. Here is an incomplete draft of such a function. You will have to replace the `#xxxxx` and use the function `payout`.

```
simulate_2P_PD <- function(strategy_p1, strategy_p2, N, payoutvector = c(5,
    3, 1, 0)) {
    total_payout <- c(0, 0)
    past_actions_p1 <- vector()
    past_actions_p2 <- vector()
    for (i in 1:N) {
        # xxxxx
    }
    return(total_payout)
}
```

2.3

Discuss with your partner what would be a good strategy. Try to implement it. Your strategy will then compete against all other strategies from other groups in a tournament.

IMPORTANT: If you have an idea but do not know how to exactly implement it, ask! Also, to make this clear: The winner of the tournament is not the strategy that does not lose (for example, always defect). Instead, it is the strategy with the highest average payoff.

If you want, you can conduct your own tournament of different strategy functions to see which one is best. In our tournament, each strategy competes against each other in a 50-round game and each competition is repeated 100 times for accuracy. Here is a draft of a tournament function that takes as input

- a list of strategy functions,
- `N`, the number of rounds per game,
- `Nsim`, the number of repetitions per pairwise N-round game,
- a payout vector,
- a Boolean variable that describes whether each strategy should perform against itself.

The function `tournament` returns an average payout matrix. The entry in row `i` and column `j` describes the average payout that strategy `i` achieved against strategy `j`.

```
tournament <- function(list_of_strategies = list(strategy1, strategy2, strategy3),
                       N = 50, Nsim = 100, payoutvector = c(5, 3, 1, 0),
                       PLAY_YOURSELF = FALSE) {

    n <- length(list_of_strategies)
    total_payouts = rep(0, n)
    av_payout_mat = matrix(0, n, n)

    for (i in 1:(n - 1 + PLAY_YOURSELF)) {
        for (j in (i + 1 - PLAY_YOURSELF):n) {
            average_payout <- ?? #Remember to average 100 repetitions
                av_payout_mat[i,j] <- ??
                av_payout_mat[j,i] <- ??
                total_payouts[i] <- total_payouts[i] + average_payout[1] /
                    (n - 1 + PLAY_YOURSELF)
                total_payouts[j] <- total_payouts[j] + average_payout[2] /
                    (n - 1 + PLAY_YOURSELF)
        }
    }
    return(av_payout_mat)
}
```

You can use the following code to evaluate how the strategies perform against each other.

```
heatmap(av_payout_mat, scale = "none", revC = TRUE, Rowv = NA,
    Colv = NA, margins = c(10, 10))
```

Also, `rowSums` and `colSums` will help you to find the best of your strategies.

2.4

Each group agrees on one strategy. Please give this strategy function the name strategy[group number], i.e. if you are group 3, you will name it `strategy3`. Send them to us via e-mail (copy-paste the function into the e-mail body) and we will have a big tournament of all strategies.

We will analyze the results of the tournament together.

2.5

If your strategy was not the best (or even then), revise your strategy and send it again for a second tournament. Is the strategy from the first tournament still the best?

2.6

Can you even conclude that the winning strategy from a tournament must actually be the best of the competing strategies? Why, why not? Watch https://www.youtube.com/watch?v=BOvAbjfJ0x0