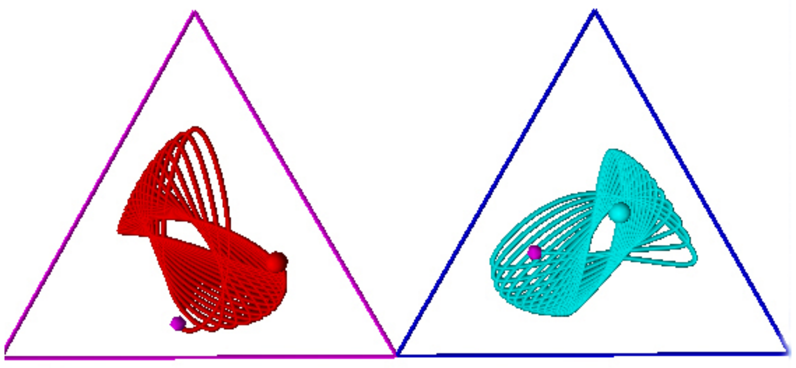
Rock-Paper-Scissors:
An Example of a Multi-agent Dynamical System



Kristin Lui

Department of Mathematics

Kristin.lui@gmail.com

Abstract: In order to study and understand multi-agent systems, I create simulations of agents playing the game Rock-Paper-Scissors for one and two agents cases. Using the differential equations from "Stability and Diversity in Collective Adaptation" explore how rewards from the environment and agents' memory loss affect the probability distribution of choosing a new action for each agent.

Introduction

I wanted to study a multi-agent system because of the complex dynamical behavior that arose from the seemly simple two agent system. Computers today not only compute for a person thousands of iterations of a system in a few seconds but also allow interaction between a system and a person in a way that paper and pencil could not accomplish. My ambitious goal was to observe the dynamical behavior of two agents in four-dimensions and I saw visual python as a perfect tool to complete such a task. Not only would the computer create and graph trajectories in a four-dimensional space for me but the computer would also allow me to test different initial conditions instantly for comparison.

Working from the paper "Stability and Diversity in a Collaborative Dynamic System", I created simulations that involved a single agent and two agents playing a game of Rock-Paper-Scissors. The simulations allowed one to change the value of parameters and also examine various initial conditions. The single agent case behaved as expected by converging to one of three possible fixed points depending on the parameters. On the other hand, the simulation involving two agents was extremely sensitive to initial conditions and changes in parameters. Graphing the trajectories in four dimensions was abandoned in favor of exploring the different limit cycles and general behavior of the two agent system.

Background

Multi-agent systems are systems in which there are several independent agents working collaboratively or against each other in order to satisfy a goal. Real world examples of multi-agent systems include flocks of birds and insect swarms. The agents in my simulations are defined as autonomous agents. These agents modify their behavior based on information they gather from the environment and from other agents. In this way, agents are not explicitly aware of the environment they are in and must learn about the environment through rewards the agents receive from interactions. Agents playing Rock-Paper-Scissors fit the conditions of a multi-agent system since the agents play independent of each other and modify their actions (playing either rock, paper, or scissors as a move) based on the rewards they receive from the environment and interactions with other agents. While each action of the individual agent is interesting, viewing the group of agents as a whole allows one to see other complex behavior and patterns. This is interesting because the agents aren't working in unison but rather are modifying their behaviors independently based on local interactions with each other.

Dynamical System

The two systems I investigated were the single agent and two agents systems of agents playing the game Rock-Paper-Scissors. The reason for studying the single agent case is to gain a better understanding of how an agent is affected by different parameters without the added complication of another agent. For an agent, I looked at the change in the agent's probability distribution with respect to time. To do this I treated each probability distribution as a vector with three components since in the game there are three possible actions. The sum of an agent's probability distribution must equal one. As a result, the trajectory of the agent's probability distribution as it changes remains within a two-dimensional triangle called a simplex (Figure 1). The rewards an agent receives is also a vector $R = (\varepsilon, -1, 1)$ where the rewards are ε for ties, -1 for loses, and 1 for wins.

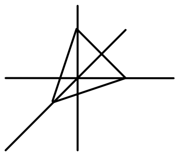


Figure 1: The triangular simplex for an agent with three components (actions).

The equation for a single agent is:

$$\frac{\dot{x}_i}{x_i} = \beta(R_i - R) + \alpha(H_i - H)$$

R_i is the reward given for the particular action $i \in (1...N)$ and $R = \sum_{n=1}^{N} x_n R_n$ which is the net reinforcement averaged over the agent's possible actions. The difference between the rewards tells the agent how the reward for the current action *i* compares to the average of the other rewards. $H_i = -\log(x_i)$ is the self-information or degree of surprise when action *i* is taken and $H = \sum_{n=1}^{N} x_n H_n$, the Shannon entropy. The difference serves to make the probability distribution uniform. The two parameters β and α control the two differences for the rewards and the self-information of each action. The adaptation rate is controlled by $\beta \in [0, \infty)$. If $\beta > 0$, the agent will increase the probability of choosing the action that led to the highest reward. In the special case of $\beta = 0$, the agent's choice distribution is unaffected by current reinforcements and chooses actions randomly. The agent's memory loss rate is controlled by $\alpha \in [0,1)$. When $\alpha > 0$, memory has less of an effect on the agent's behavior by giving past reinforcements less effect than current reinforcements. In general, the probability distribution becomes more uniform. In the special case of $\alpha = 1$, there is total memory loss and the probability distribution is uniform and converges to the Nash equilibrium of $\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$. The Nash equilibrium represents the point at which the agent chooses actions that yield the highest rewards for all parties which are the single agent and the environment. In the second special case of $\alpha = 0$, the agent has perfect memory and the agent's behavior will be governed by $\beta(R_i - R)$.

The equations for the two agent case are:

$$\frac{\dot{x}_i}{x_i} = \beta_X[R_i^X - R^X] + \alpha_X[H_i^X - H^X]$$
$$\frac{\dot{y}_j}{y_j} = \beta_Y[R_j^Y - R^Y] + \alpha_Y[H_j^Y - H^Y]$$

Here the equations are the same as the single agent case. One difference is the rewards scheme for two agents. There are two matrices, one for each agent, which describe the rewards based on the agents' interaction with each other. $A = \begin{bmatrix} \varepsilon_X & -1 & 1 \\ 1 & \varepsilon_X & -1 \\ -1 & 1 & \varepsilon_X \end{bmatrix}$ and $B = \begin{bmatrix} \varepsilon_Y & -1 & 1 \\ 1 & \varepsilon_Y & -1 \\ -1 & 1 & \varepsilon_Y \end{bmatrix}$ where ε_X and ε_Y are ties for Agent 1 and Agent 2 respectively. Additionally, another difference is how the *R*'s and *H*'s are calculated and those equations can be found in the Sato paper (equations ….). For two agents, the main dynamical behavior occurs in four-dimensional space since the state space for the two agents is two-dimensional; the product of the state spaces gives four dimensions.

Methods

I used visual python to simulate the change in probability distribution for an agent (single-agent case) or agents (two-agent case). For all simulations, I used the fourth order Runge-Kutta integration method to solve the differential equations. In the single-agent case, I allowed the user to vary the alpha and beta values during the simulation. I believe the instant feedback aids in the user's understanding of how those parameters affect where the distributions converge. The simulation for two-agents also allows the user to vary the alpha and beta parameters for Agent 1 and Agent 2. Even though the epsilons cannot be varied, additional sliders can be added to the simulation. A clear button is available so that the transient trajectories can be cleared from the screen. Transient trajectories are included in the display because the initial behavior of the agents is also interesting to watch. Since there are so many parameter values to choose from, I decided to use specific parameters from the paper Sato, et al. so that the system observed would be chaotic. These parameters are $\alpha_X = 0.0198$, $\beta_X = 1.0$, $\varepsilon_X = 0.5$, $\alpha_Y = 0.01$, $\beta_Y = 1.0$, and $\varepsilon_Y = -0.3$. With these values, the simulation's main function was to show the trajectories of the two agents with varying initial conditions.

Results

There were three main outcomes from the single agent case which were convergence at the vertex, inner simplex, and center of the simplex. The trajectory converged to a vertex of the simplex when $\alpha = 0$ and $\beta > 0$. The vertices of the triangle are points at which the agent is playing only one action 100% of the time. Converging to one of these vertices means that the agent is sensitive to the current rewards it receives and modifies the probability distribution to maximize the current rewards (Figure 2). When the trajectory converged to a point inside the simplex, the agent had some memory loss since $\alpha > 0$ (Figure 3). This means that the agent's probability distribution was becoming uniform but rewards from the environment prevented the trajectory from converging to the Nash equilibrium. Finally, there was total memory loss when $\alpha = 1$ and the trajectories became uniform and converged to the center of the simplex (Figure 4). In addition to setting the parameter values before running the simulation, the user can change the values during the run also. The result is a funny looking curve (Figure 5) but while changing the parameters the user can see and understand how convergence is affected by β and α.

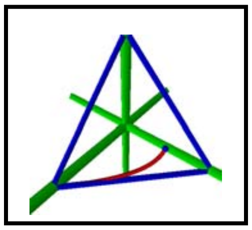


Figure 2: Convergence at the simplex's vertex shows the agent reacting strongly to current rewards and going towards a probability distribution of playing one action 100% of the time.

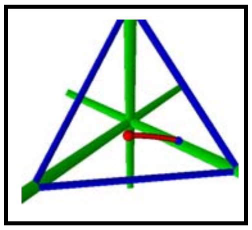


Figure 3: Convergence inside the simplex indicates some memory loss ($\alpha > 0$)

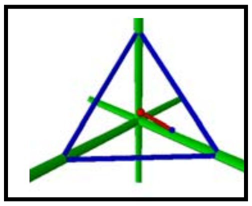


Figure 4: The agent converges to the center of the simplex due to complete memory loss.

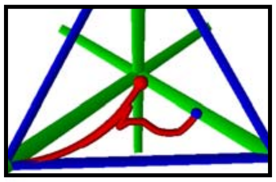


Figure 5: The trajectory can be manipulated during the simulation by changing the parameter values.

In the two agents simulation, convergence was in the form of limit cycles for both agents. Limit cycles were usually in the form of a triangle that was within the simplex while other cycles were trefoil patterns or almost circular orbits that occurred in the middle of the simplex. The transient phase of the trajectories would sometimes look random but after using the "clear" button in the simulation, the true behavior would be revealed. Trajectories were found to end in a limit cycle of either a triangle or an irregular "spirographic" shape that creates an interior space (Figures 6 and 7). It is not known what other limit cycle shapes there are if any others exist.

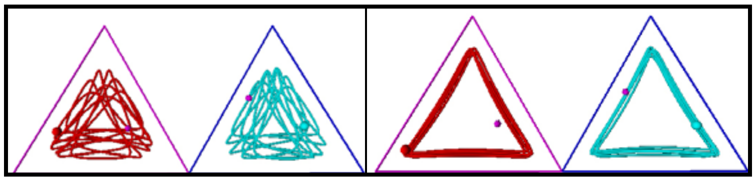


Figure 6: An initial condition that started chaotic (left) and ended in a triangular limit cycle (right).

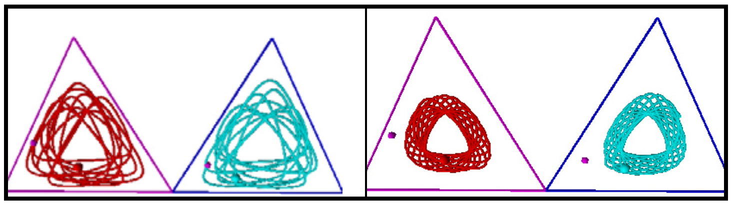


Figure 7: A different initial condition with random behavior (left) but the trajectory settles down into a "spirographic" cycle (right)

Other conditions start with on the limit cycles such as the irregular "spirographic" shape (Figure 8), the triangular orbit (Figure 9) and an almost circular orbit (Figure 10). The irregular "spirographic" shape repeats indefinitely in the interior of the simplex, is not perfectly circular, and has a triangular interior space. The triangular orbit seems to develop from initial conditions that lie near the edges of the simplex. Finally the almost circular orbit appears when the initial conditions are near the center of the simplex and are exactly the same in value for both agents. I found that when the agents had the same value for their initial conditions, the trajectories would slowly spiral to the almost circular orbit inside the simplex.

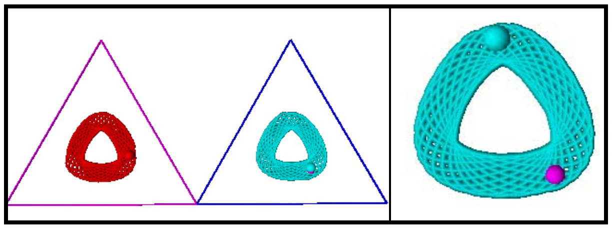


Figure 8: Trajectories that started in the irregular "spirograph" (left). An interior triangular space can be seen in the close-up of one of the trajectories (right).

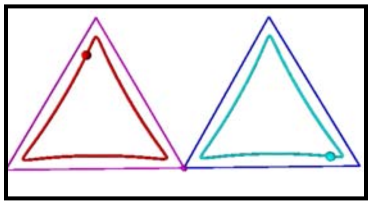


Figure 9: A triangular orbit.

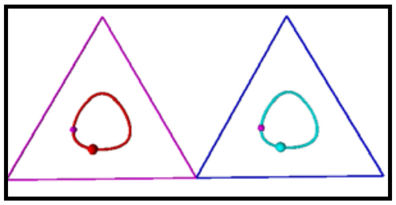


Figure 10: An almost circular orbit inside the simplex.

Another set of interesting initial conditions are those that started on one of the simplex's edges (Figure 11). In this case, the trajectories stay mostly on the edges of the simplex showing that the agents are playing a sub-game by using only two of the three actions. However, after a transient period, the trajectories settle into a triangular limit cycle within the simplex.

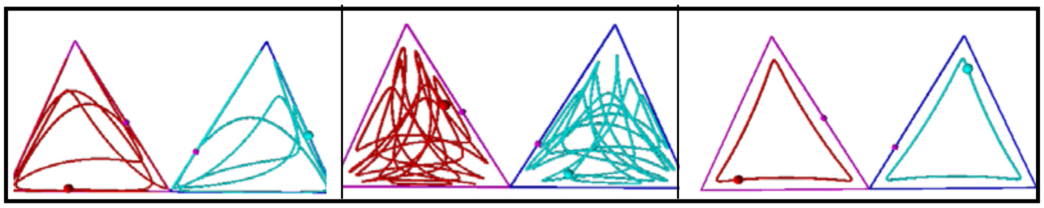


Figure 11: Initial conditions on an edge of the simplexes. The trajectories travel from one edge to another (left), then the trajectories become random (middle), and finally the trajectories settle into a triangular orbit (right).

Conclusion

Although simple, the simulation for the single agent case allowed one to explore how the parameters β and α affect the dynamical behavior of an agent enabling the user to understand how memory loss and rewards affect convergence. For the two agents case, even though the parameters weren't varied, as in the one agent case, partial understanding of the complicated system was achieved by observing different initial conditions. By changing the initial conditions, several different limit cycles were found. Also the use of the "clear" button allowed one to look beyond the interesting transient state to the actual limit cycles. The end limit cycles of the agents show that with some memory loss, the agents settle into a pattern of switching between the three actions which results in the triangular shapes of the cycles. A more systematic and mathematical approach could be used to classify and predict the limit cycles and exploration of how varying the parameters while the simulation is running would yield more information about how agents adapt to one another.

Bibliography

Y. Sato, E. Akiyama, and J. P. Crutchfield, "Stability and Diversity in Collective Adaptation," Journal of Theoretical Biology (2004) submitted.