JEDy: A Julia package for Evolutionary Dynamics

Nikolas M. Skoufis
Supervisor: Julián García

October 24th, 2014

Abstract

Evolutionary dynamics is the branch of biology concerned with studying how populations of invidivuals with inherited traits change over time. Performing simulations and calculations related to game theoretic views of evolutionary dynamics can be computationally intensive. In this project we have developed a library for performing common computational evolutionary dynamics calculations in the high performance scientific computing language Julia.

1 Introduction

Computational evolutionary dynamics studies the way that populations evolve when subject to inheritance of traits and mutation. The theoretical aspects of computational evolutionary dynamics are rooted in the mathematical discipline of game theory, however it is often useful to verify results with simulations. As the computations involved are generally repetitive and similar across various scenarios, it becomes useful to create packages of commonly used functions.

 Packages for performing evolutionary dynamics calculations exist for languages such as Python [2] an Mathematica [5]. however these languages are notorious for being orders of magnitude slower than C or FORTRAN. However low level, high performance languages like C or FORTRAN do not offer the nice syntax and useful data strucures offered by higher level languages. In order to combine the performance of low level languages with the syntax of high level languages, we chose to write our package in Julia [1].

 Julia is a relatively new language designed for high performance scientific computing. Julia is syntactically inspired by Matlab and Python, but it remains fast. It is dynamically typed, uses multiple dispatch, is designed for parallelism and distributed computing, and benchmarks have shown that it can achieve near C speeds. Julia also has nice features like package management, access to IPython's notebook capabilities (for embedding rich text alongside code and figures), and the ability to call C and Python code. Julia is still under heavy development, and this created some problems during the development of this package.

2 Background

In order to understand what is involved in writing a package to perform computational evolutionary dynamics calculations, it will be useful to understand some of the game theory underpinnings of the calculations. While developing JEDy, we used the problem of the iterated prisoners dilemma in order to develop and test our package. Below we will present the problem and the mathematics behind simulating the system and solving for various quantities.

2.1 Iterated prisoner's dilemma

The prisoner's dilemma is a game involving two players. Each player is able to either cooperate or defect, and the four different combinations of these moves provides a different payoff to each player. If both players cooperate, they each receive payoff R. If both players defect, they each receive a lower payoff P. However if one player defects and the other cooperates, the cooperator receives the lowest payoff S, while the defector receives the highest payoff T. Therefore, we have payoffs such that $T < R < P < S$.

If we play the prisoner's dilemma multiple times, we have the ability to choose strategies. One strategy is to always cooperate (ALLC). Another possible strategy would be to allways defect (ALLD). The best strategy know is tit-for-tat (TFT) (or a minor variation of TFT) [4], where the player cooperates in the first game and then mimics the move that their opponent plays in the last game in subsequent games. Since TFT has some additional complexity over ALLC or ALLD, it is natural to introduce a complexity cost for this strategy. If we have a finite number of rounds m and a complexity cost c that reduces the payoff for TFT, we can construct a matrix which gives the payoff for each pairing of strategies.

$$
\begin{array}{c}
\begin{array}{ccc} \text{ALLC} & \text{ALLD} & \text{TFT} \end{array} \\
\begin{array}{c} \text{ALLC} \\ \text{ALLD} \\ \text{TFT} \end{array}
\left(
\begin{array}{ccc}
Rm & Sm & Rm \\
Tm & Pm & T + P(m-1) \\
Rm - c & S + P(m-1) - c & Rm - c
\end{array}
\right)
\end{array}
$$

2.2 Reproduction, death and mutation: the Moran process

In studying the iterated prisoner's dilemma, we used the Moran process to model how populations of individuals evolve over time while playing the game. The Moran process is a type of birth-death process, where at each timestep the following steps occur in order:

1. An individual is chosen to reproduce. In reality this involves choosing a strategy to reproduce with probability proportional to their fitness (to be defined later).

4

2. The offspring of the reproducing individual may mutate to one of the other strategies with equal probability μ.

3. An individual is chosen to die. Each individual is chosen equiprobably.

The fitness function used in the Moran process is frequency dependant. Given a population with N individuals, with strategy populations x_1, x_2, x_3, and with payoff matrix A with elements a_{ij}, the fitness function can be defined as

$$f(i) = \frac{\sum_{j=0}^{3}(a_{ij}x_j) - a_{ii}}{N - 1}$$

for $i \in 1, 2, 3$. This is essentially the average payoff given that games are played against a random opponent. We subtract a_{ii} and divide by $N - 1$ because the player cannot play against themselves.

2.3 Intensity of selection

Another quantity of interest in computational evolutionary dynamics is the intensity of selection. The intensity of selection is a paramter which controls how much of a role fitness plays in selection. If we define p to be the payoff from the game and w to be the intensity of selection, we can transform the payoff values either linearly with

$$\pi(p) = 1 - w + wp, \quad w \in [0, 1]$$

or exponentially with

$$\pi(p) = e^{wp}, \quad w \in [1, \infty).$$

The benefit of transforming exponentially is that this allows us to deal with negative payoffs, where as the linear intensity of selection map cannot deal with negative payoffs.

As the intensity of selection approaches 0, all mapped payoffs approach 1. This creates a Moran process which is essentially a random walk between states, since at each timestep, each strategy is equally likely to be chosen to reproduce. As the intensity of selection approaches 1, the mapped payoffs approach their unmapped values, and as such, play a large role in selection.

Evolutionary dynamicists are often interested in the effect of intensity of selection on stationary distribution, which we will discuss in the next section.

2.4 Fixation, transition matrices and calculating stationary distribution

Another quantity of interest in evolutionary dynamics is the stationary distribution. The stationary distribution represents the long term proportions of time spent in the homogenous states, that is, the states where the population consists of only one strategy.

In order to calculate the stationary distribution, one must first derive the transition matrix, and hence, the probability of fixation.

The probability of fixation is the probability that, given a starting state where all individuals are of strategy A (the dominant strategy) but for one individual of strategy B (the mutant strategy), the population eventually transitions to a state where all individuals are of the mutant strategy. If we assume that the time between mutations is much larger than the time it takes for the population to fixate on one strategy, we can use this to derive a transition matrix which gives the probability of transitioning between different states.

A lengthy derivation of the fixation probability can be found in [6], but the relevant details are that the fixation probability takes the form

$$\mathbb{P}(\text{fixation}, i \text{ mutants}) = \frac{1 + \sum_{k=1}^{i-1} \prod_{j=1}^{k} \gamma_j}{1 + \sum_{k=1}^{N-1} \prod_{j=1}^{k} \gamma_j}$$

where

$$\gamma_j = \frac{T_j^-}{T_j^+}$$

and where T_j^- and T_j^+ represent the probability that the state where the mutant population is of size j decreases by one or increases by one respectively.

T_j^- and T_j^+ are given by

$$T_j^- = \mathbb{P}(\text{dominant pop reproduces}) \times \mathbb{P}(\text{mutant dies})$$
$$T_j^+ = \mathbb{P}(\text{mutant pop reproduces}) \times \mathbb{P}(\text{dominant dies}).$$

If we set i equal to 1 in the equation above for fixation probability, and then enumerate all the possible combinations of dominant and mutant strategies, we can obtain the transition matrix for the process. The fixation probabilities along the diagonal (where dominant and mutant strategies are the same strategy) are chosen so as to keep the matrix stochastic. This gives us the probability that we will transition between states if a mutation happens to occur while we are in a fixated state. Since this is simply a markov chain process from elementary probability, in order to observe the long term

distribution of states we can take the eigenvector that corresponds to the eigenvalue of 1, scale it so that it sums to 1, and thus obtain the stationary distribution for the system.

3   JEDy: Julia for Evolutionary Dynamics

The main goal of this project was to kickstart development of a package for Julia for performing the sorts of calculations outlined in the previous section. My task was to develop the methods for dealing with evolutionary dynamics over finite populations, while my colleague Jesse Duffield developed the methods for infinite populations. As of writing, we have decent coverage of the methods mentioned in the previous section, and we're able to visualise these results.

The goal for this stage of the package's development was to be able to replicate the results found in a paper by Imhof et. al. [3] which studied the Prisoner's dilemma. The heatmap shown in figure 1 is similar (though not as nicely formatted) to the diagram found in Imhof et. al., and displays the same trend found in the paper, namely, that the population spends most of its time in the homogenous TFT state.
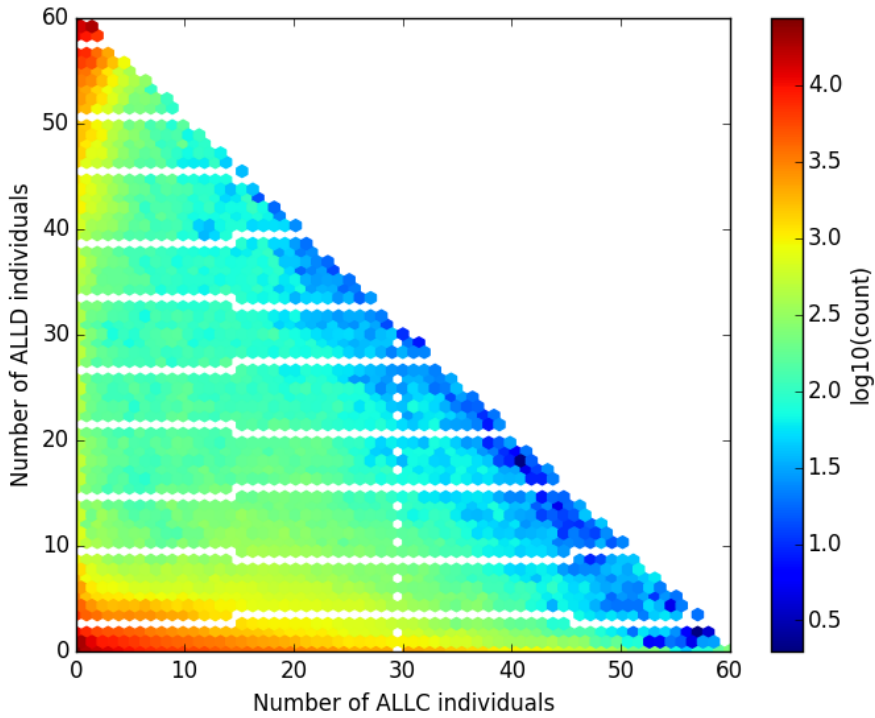

Figure 1: A heatmap of state distributions in the iterated prisoners dilemma using a Moran process simulation. Parameters are 1×10^6 iterations at $N = 60, T = 5, R = 3, P = 1, S = 0.1, m = 10, c = 0.8$ with mutation rate $\mu = 1 \times 10^{-2}$ and with linear intensity of selection and intensity of selection 1. The bottom left corner represents the states when the population is mostly TFT. The results here fairly closely match those presented by Imhof et. al. The white lines are artifacts of the plotting process.

We can further confirm that our results match those of Imhof et. al. by studying the

plot of the timeseries found in figure 2. This plot also shows that the population spends most of its time in the homogenous TFT state. Furthermore, it shows tha the population tends to oscillate from homogenous TFT, to homogenous ALLC, to homogenous ALLD, then back to homogenous TFT. This also matches the results given in Imhof et. al., and also shows the low residence time in the homogenous ALLC state they reported.
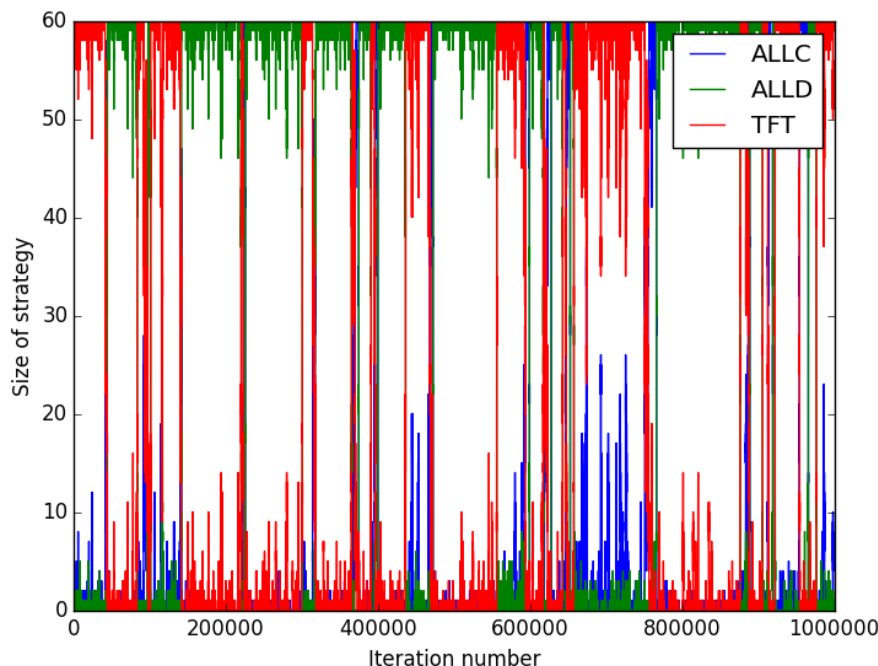

Figure 2: A time series of the iterated prisoner's dilemma using a Moran process simulation. Parameters are 1×10^6 iterations at $N = 60, T = 5, R = 3, P = 1, S = 0.1, m = 10, c = 0.8$ with mutation rate $\mu = 1 \times 10^{-3}$ and with linear intensity of selection and intensity of selection 1. The population spends most of its time in the TFT state, followed closely by the ALLD state.

Finally, a plot of the intensity of selection against the stationary distribution in figure 3 shows the expected behaviour:

- When intensity of selection is close to 0, the stationary distribution is evenly split among the strategies

- When intensity of selection is close to 1, the TFT strategy dominates.

In addition to these functions dealing with finite populations, JEDy can also deal with simulations involving infinite populations. This portion of the package was handled by my colleage Jesse Duffield. In short, the infinite population functios are able to simulate the prisoner's dilemma using the replicatior and replicator-mutator equations [3], as well as visualise the results using phase portraits and vector field diagrams.
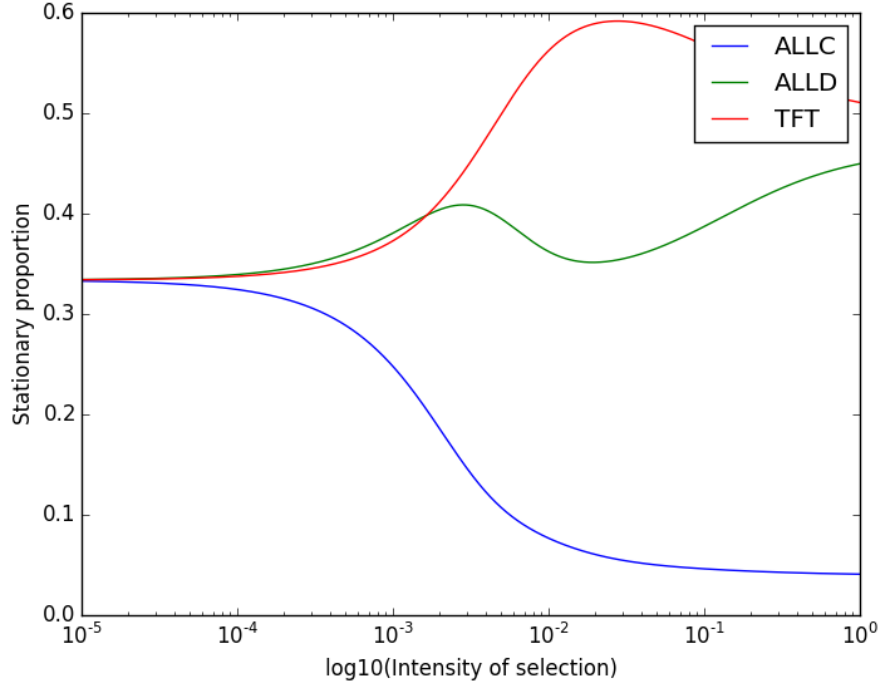
9

Figure 3: A plot of stationary distribution against intensity of selection for the iterated prisoner's dilemma using a Moran process simulation. Parameters are $N = 60, T = 5, R = 3, P = 1, S = 0.1, m = 10, c = 0.8$ with mutation rate $\mu = 1 \times 10^{-3}$ and with linear intensity of selection and intensity of selection between 1×10^{-5} and 1 in 1000 setps distributed logarithmically. The stationary distribution is evenly distributed among the strategies when intensity of selection is close to zero, as expected. When the stationary distribution is close to one, the stationary distribution is dominated by the TFT state, also as expected.

4 Conclusion and future directions

Over this semester we have succeeded in developing a basic package for performing computational evolutionary dynamics calculations in Julia. We are able to simulate the Moran process and the replicator-mutator equation and display the results of our simulationas, as well as being able to compute some useful quantities such as stationary distributions. We have found Julia to be a user friendly language to program in, and it has many advanced features, despite offering some resistance at times due to the fact that it is still under heavy development.

One point of note is that there is currently no docstring convention for Julia (though one is being discussed). As such the documentation of our code is far from comprehensive, and this something that we would hope to address in the future.

Other future goals for JEDy include

- Make JEDy an official Julia package by registering it at the Julia Github repository.

- Add additional functionality for performing computations including different simulation processes.

- Migrate the data visualisation functionality from PyPlot (a Julia wrapper around the mature Python library matplotlib) to a native Julia package such as Gadfly.

- Clean up the code to make it more readable and more modular.

- Allow for the use of more general games, ie. games with more than two players.

- Optimise the functions for memory usage and performance.

References

[1] Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and other contributors. The julia language. http://julialang.org, 2014.

[2] Julian Garcia. Evolutionary dynamics with python. https://github.com/juliangarcia/pyevodyn, 2014.

[3] Lorens A Imhof, Drew Fudenberg, Martin A Nowak, and Robert M May. Evolutionary cycles of cooperation and defection. *Proceedings of the National Academy of Sciences of the United States of America*, 102(31):10797–10800, 2005.

[4] Martin Nowak and Karl Sigmund. A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game. *Nature*, 364(1):56–58, 1993.

[5] Bill Sandholm, Emin Dokumaci, and Francisco Franchetti. Dynamo: Diagrams for evolutionary game dynamics. http://www.ssc.wisc.edu/~whs/dynamo/, 2014.

[6] Arne Traulsen and Christoph Hauert. Stochastic evolutionary game dynamics. In Heinz Georg Schuster, editor, *Reviews of nonlinear dynamics and complexity*, volume 2. Wiley-VCH, 2009.