Using DyPy, A Powerful Python Library for Simulating Evolutionary Dynamics of Matrix-Form Games

Andrew Ferdowsian

February 10, 2016

1 Introduction

In this white paper, I demonstrate how to use DyPy,[1] a Python library that permits users to easily run simulations of three evolutionary dynamics: Replicator, Wright-Fisher, and Moran[2] for any matrix form game. The library is flexible: it can be used to analyze a game with any number of players and strategies. It is has been parallelized and optimized, to ensure simulations run relatively fast. It has a variety of built-in visualizations that allow the user to easily present the results of her simulations. Most importantly, it is easy to use: most simulations require only 20-100 lines of code, which can be written without previous knowledge of Python.

DyPy is open source. It is hosted on Github[3]. Improvements through pull requests are welcome. Suggestions for additional functionality are also encouraged. A wiki is included in the Github repository; it provides detailed documentation for each command in the library.

To demonstrate the library's functionality, I have simulated and presented the evolutionary dynamics of some well-known matrix-form games: a Costly Signaling game, the Hawk Dove Bourgeois game, and the Envelope Game.

2 Costly Signaling

The costly signaling model (Spence 1973) was originally developed to explain why labor market participants might invest in expensive and not-especially-useful higher

[1] DyPy was originally developed by Eric Lubin, then expanded by myself, with the guidance of my colleagues at the Program for Evolutionary Dynamics.

[2] Note that I assume the reader is familiar with these evolutionary dynamics throughout. For a refresher, see Nowak (2006).

[3] `https://github.com/aaandrew152/dynamics_sim`

education. The intuition is as follows. Suppose some employees are more productive than others. However, when interviewing, employers cannot tell a productive employee from an unproductive one, and productive employees cannot credibly communicate that they are productive. Employees, can, however, invest in higher education, which, we assume, confers no productivity benefits. If productive employees are also better students or otherwise find it less costly to invest in a degree, then, even though the degree itself doesn't confer any productivity benefits, in the "separating" equilibrium, the productive employees will invest in the degree, unproductive employees will not, and employers will only hire employees with degrees. The degree distinguishes productive employees from unproductive ones.

Variations of the model have been widely applied both in settings where individuals deliberately choose whether to signal, and also to settings where they do not. Perhaps the most famous example of the latter is to explain long ornamental tails in birds like the peacock (Zahavi 1975). Other examples include humans' sense of aesthetics (Hurst, Charles and Roussanov 2008) and religious rituals (Irons 1996). To apply the model in these settings, researchers rely on evolutionary dynamics, rather than standard assumptions on rationality, to justify the use of the model. That is, rather than assume that agents deliberately choose whether to signal, they assume that they evolve, learn, or imitate their strategies (whether to signal; whether to rely on signals when matching), and that strategies that are more successful become more common over time. Then, as I will demonstrate momentarily, the costly signaling equilibrium often emerges (Hoffman and Yoeli 2015).

I will now replicate the Costly Signaling analysis (Hoffman and Yoeli 2015). In their variation of the game, there are two players; the sender and the receiver. (1) The sender is of two possible types: low with probability p, or high with probability $1-p$. (2) The sender must decide a level of signal s to send from $\{0, 1, 2, 3\}$. (3) The receiver, after observing the signal, decides whether to accept or reject the sender.

The sender's receives $-c \cdot s$ in payoff regardless of the receiver's decision, and additionally receives a if accepted. The receiver's payoff is 0 if they reject the sender, b if they accept a high sender, and $-d$ if they accept a low sender, where $a, b, c, d > 0$. If for some s, $-c_l \cdot s + a < 0$ and $-c_h \cdot s + a \geq 0$ it is a Nash Equilibrium for the high sender to send a signal s, the low sender to send no signal, and the high receiver to only accept a receiver who sends a signal s.

Although this is the equilibrium of interest which henceforth we shall call the separating equilibrium, additional equilibria exist. We use the dynamics to see if

the separating equilibrium emerges a significant proportion of the time.

To prepare the game[4] the following 43 lines of code are used:

```
from games.game import Game

class Costly_Signaling(Game):
    DEFAULT_PARAMS = dict(lCost=3, hCost=1, a=5, aHigh=10, aLow=-10, lProp=2, hProp=1)
    PLAYER_LABELS = ('Low Sender', 'High Sender', 'Receiver')
    STRATEGY_LABELS = (('No', 'Low', 'Medium', 'High'),
                       ('No', 'Low', 'Medium', 'High'),
                       ('Accept all','Accept Low','Accept Medium','Accept High','Reject All'))

    def __init__(self, lCost, hCost, a, aHigh, aLow, lProp, hProp, equilibrium_tolerance=0.2):
        lProp, hProp = lProp/(lProp+hProp), hProp/(lProp+hProp)
        lProp /= 2
        hProp /= 2

        payoff_matrix_p1 = [[[0 for x in range(5)] for x in range(4)] for x in range(4)]
        for i1, lowSenderStrat in enumerate(payoff_matrix_p1):
            for i2, highSenderStrat in enumerate(lowSenderStrat):
                for i3, payoff in enumerate(highSenderStrat):
                    if i1 >= i3:
                        payoff_matrix_p1[i1][i2][i3] += a
                    payoff_matrix_p1[i1][i2][i3] -= lCost * i1

        payoff_matrix_p2 = [[[0 for x in range(5)] for x in range(4)] for x in range(4)]
        for i1, lowSenderStrat in enumerate(payoff_matrix_p1):
            for i2, highSenderStrat in enumerate(lowSenderStrat):
                for i3, payoff in enumerate(highSenderStrat):
                    if i2 >= i3:
                        payoff_matrix_p2[i1][i2][i3] += a
                    payoff_matrix_p2[i1][i2][i3] -= hCost * i2

        payoff_matrix_p3 = [[[0 for x in range(5)] for x in range(4)] for x in range(4)]
        for i1, lowSenderStrat in enumerate(payoff_matrix_p1):
            for i2, highSenderStrat in enumerate(lowSenderStrat):
                for i3, payoff in enumerate(highSenderStrat):
                    if i1 >= i3:#Low sender accepted
                        payoff_matrix_p3[i1][i2][i3] += aLow * lProp
                    if i2 >= i3:
                        payoff_matrix_p3[i1][i2][i3] += aHigh * hProp

        payoff_matrix = [payoff_matrix_p1, payoff_matrix_p2, payoff_matrix_p3]
        player_dist = (lProp, hProp, 1/2)
        super(Costly_Signaling, self).__init__(payoff_matrices=payoff_matrix, \
         player_frequencies=player_dist, equilibrium_tolerance=equilibrium_tolerance)
```

Once the game has been created, a variety of simulations can be chosen from,

[4]See the wiki at `https://github.com/aaandrew152/dynamics_sim/wiki` for detailed documentation for each command

ranging from a plot of a single run to wire frame plots, which change two variables over many runs. For example:

```
def test_single_simulation(self):
    s = GameDynamicsWrapper(Costly_Signaling, WrightFisher)
    s.simulate(num_gens=50000, graph=dict(area=True))
```

This simulates 50,000 generations of Wright-Fisher dynamics and outputs the following graphs, these can be saved as .png files:

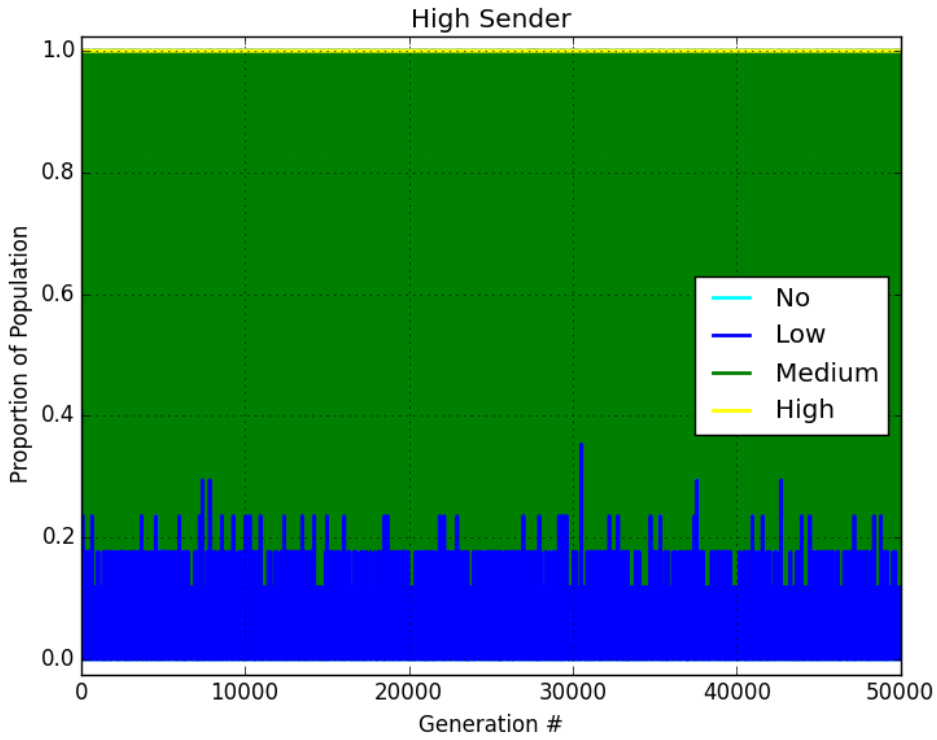

Figure 1: Proportion of High Senders Sending Each Signal

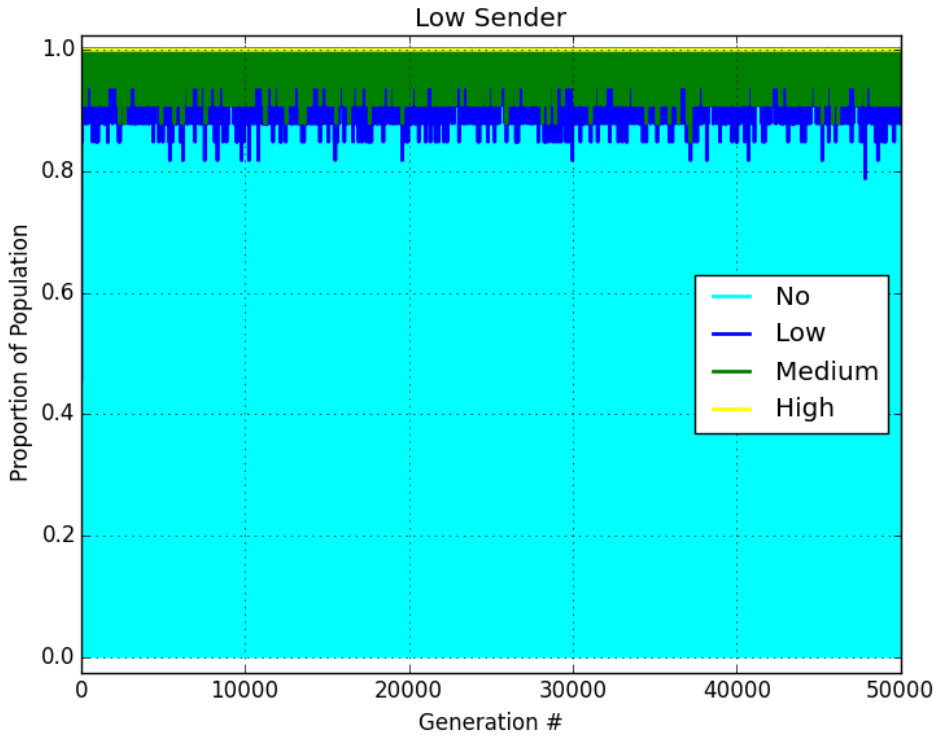

Figure 2: Proportion of Low Senders Sending Each Signal

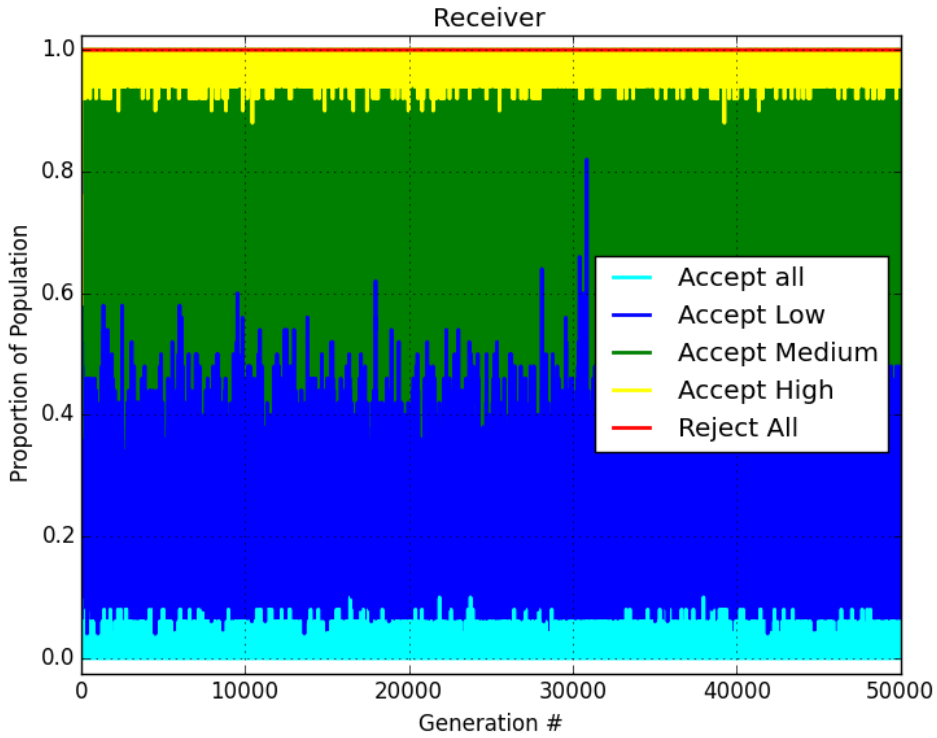

Figure 3: Proportion of Receivers Accepting Each Signal

Figure 4: Frequency of Strategies Over 50,000 Generations in a Single Run of the Wright-Fisher Dynamic

As we see from the first two graphs, the high sender nearly always sends a medium signal, while the low sender sends no signal. The receiver accepts those who send signals while rejecting those who do not send signals. The dynamics demonstrate that the separating Nash Equilibrium is frequently reached.

3 Hawk-Dove-Bourgeois

The Hawk-Dove-Bourgeois model was created by Maynard-Smith (1973) to explain a puzzling behavior in animals: Animals which arrive first at a resource such as food, territory, or a mate fiercely defend it against intruders, regardless of an apparent disparity in strength.

The game has also been used to explain our sense of property rights, and why it depends on "uncorrelated asymmetries" like who first possessed an item (DeScioli and Wilson 2011, Hoffman, Yoeli and Navarrete 2016).

There are two players in the Hawk Dove Bourgeois game. They are contesting an object worth v. Both players simultaneously choose to fight (Hawk), concede (Dove), or fight if they have arrived first (Bourgeois). If they both fight they pay a cost $c > v$ and receive the object with probability $1/2$. It is randomly determined who arrives first, with probability $1/2$.

There exist three Nash Equilibria in this game: $(H, D), (D, H)$, and (B, B). Utilizing evolutionary simulations we find that the equilibrium which always emerges in a single population is (Bourgeois, Bourgeois).

The Hawk-Dove-Bourgeois game is symmetric. That is, all players have the same strategies and payoff matrices. For such games, we use a subclass of the game class. Additionally the library allows for classifications of expected equilibria (see the *classify class* method):

```
from games.game import SymmetricNPlayerGame

class HawkDoveBourgeois(SymmetricNPlayerGame):
    DEFAULT_PARAMS = dict(v=30, c=60)
    STRATEGY_LABELS = ('Hawk', 'Dove', 'Bourgeois')

    def __init__(self, v, c):
        payoff_matrix = (((v - c) / 2, v, 3 * v / 4 - c / 4),
                         (0, v / 2, v / 4),
                         ((v - c) / 4, 3 * v / 4, v / 2))

        super(HawkDoveBourgeois, self).__init__(payoff_matrix, 2)
```

```
    @classmethod
    def classify(cls, params, state, tolerance):
        threshold = 1 - tolerance

        if all(x for x in [state[0][0], state[1][1]]) >= threshold:
            return 0#Hawk Dove
        elif all(x for x in [state[0][1], state[1][0]]) >= threshold:
            return 1#Dove Hawk
        elif all(x for x in [state[0][2], state[1][2]]) >= threshold:
            return 2#Bourgeois Bourgeois
        else:
            return super(HawkDoveBourgeois, cls).classify(params, state, tolerance)
```

To determine which equilibrium is most likely, we run 1000 iterations of the simulation and report the proportion of the time each equilibrium occurs:

```
def test_many_simulation(self):
    s = GameDynamicsWrapper(HawkDoveBourgeois, WrightFisher)
    print(s.many_single_population(num_iterations=1000, num_gens=1000))
```

This command returns the results results as a text output with the proportion of each equilibrium:

```
{'Hawk Dove': 0, 'Bourgeois': 0.9714, 'Dove Hawk': 0}
```

4 Envelope Game

The Envelope Game was developed to explain why we attend to thoughts and not just actions, when assessing the moral worth of others' good deeds (Hoffman, Yoeli and Nowak 2014).

The Envelope Game is a repeated game. In each stage: (1) Nature randomly determines the temptation to defect. It is high with probability p, and low otherwise. This realization of the temptation is placed in a sealed envelope. (2) Player 1 first decides whether to open the envelope and learn of the benefit from defection. (3) Then, player 1 chooses whether to cooperate. Player 2 observes whether player 1 opened the envelope and whether she cooperated. (4) Player 2 then decides whether to continue the game. If she decides to continue, the game repeats with probability ω. Otherwise, the game ends, and players receive payoffs $(0, 0)$ forever after.

It is an equilibrium for player 1 to cooperate without looking and player 2 to end if player 1 looks or defects. We call this the "cooperate without looking" equilibrium. There are other equilibria of the game. To determine whether or not it will emerge

a high proportion of the time, we simulate it over multiple iterations while varying ω.

We generate the game as follows:

```
from games.game import Game

class CWOL(Game):
    DEFAULT_PARAMS = dict(a=1, b=1, c_low=4, c_high=12, d=-10, w=0.895, p=0.51, player1_prop=0.5)
    PLAYER_LABELS = ('Player 1', 'Player 2')
    STRATEGY_LABELS = (('CWOL', 'CWL', 'C if Low', 'All D'),
                       ('Exit if Look','Exit if Defect','Exit if defect when low','Always Exit'))
    EQUILIBRIA_LABELS = ('CWL', 'CWOL', 'ONLY L', 'All D')

    def __init__(self, a, b, c_low, c_high, d, w, p, player1_prop, equilibrium_tolerance=0.2):
        matrix_p1 = ((a / (1 - w), a / (1 - w), a / (1 - w), a),
                     (a, a / (1 - w), a / (1 - w), a),
                     (a * p + c_high * (1 - p), (a * p + c_high * (1 - p)) / (1 - p * w), \
                       a * p + c_high * (1 - p), a * p + c_high * (1 - p)),
                     (c_low * p + c_high * (1 - p), c_low * p + c_high * (1 - p), \
                       (c_low * p + c_high * (1-p)) / ((1-w * p)), c_low * p + c_high * (1-p)))

        matrix_p2 = ((b / (1 - w), b / (1 - w), b / (1 - w), b),
                     (b, b / (1 - w), b / (1 - w), b),
                     (b * p + d * (1 - p), (b * p + d * (1 - p)) / (1 - p * w), \
                        (b * p + d * (1 - p)) / (1 - w), b * p + d * (1 - p)),
                     (d, d, d / ((1 - w * p)), d))

        payoff_matrix = [matrix_p1, matrix_p2]
        player_dist = (player1_prop, 1 - player1_prop)
        super(CWOL, self).__init__(payoff_matrices=payoff_matrix, \
          player_frequencies=player_dist, equilibrium_tolerance=equilibrium_tolerance)

    @classmethod
    def classify(cls, params, state, tolerance):
        threshold = 1 - tolerance

        if state[0][1] >= threshold:
            return 0#Cooperate with looking
        elif state[0][0] >= threshold:
            return 1#Cooperate with out looking
        elif state[0][2] >= threshold:
            return 2#Only cooperate when low
        elif state[0][3] >= threshold:
            return 3#All D
        else:
            return super(CWOL, cls).classify(params, state, tolerance)
```

Once the game has been prepared we generate our simulation:

```
def test_vary_one(self):#Simulates while changing a single variable over time
```

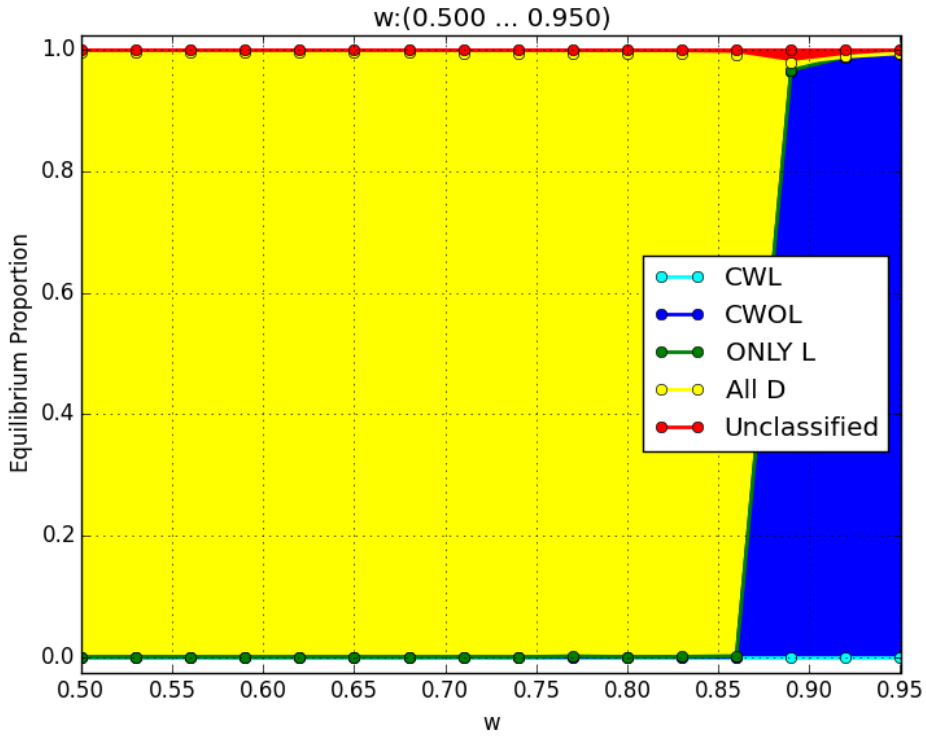

Figure 5: Frequency of the Cooperate Without Looking Equilibrium, as a Function of the Parameter ω

```
s = VariedGame(CWOL, WrightFisher)
s.vary_param('w', (0.5, 0.95, 15), num_gens=5000,  \
num_iterations=50, graph=dict(area=True))
```

Immediately after ω (the chance of a repeated interaction) grows sufficiently large, the equilibrium switches from the one in which players always defect to cooperate without looking.

5 Appendix

The library contains a variety of graphing methods, which were not all shown above. A few are included below for the Costly Signaling game to showcase the options DyPy provides.

5.1 Contour

First an example of a contour graph where two parameters are varied. We simulate the Hawk Dove Bourgeois game varying both v and c over 50 steps for 50 iterations. We included a line to show when $v > c$.

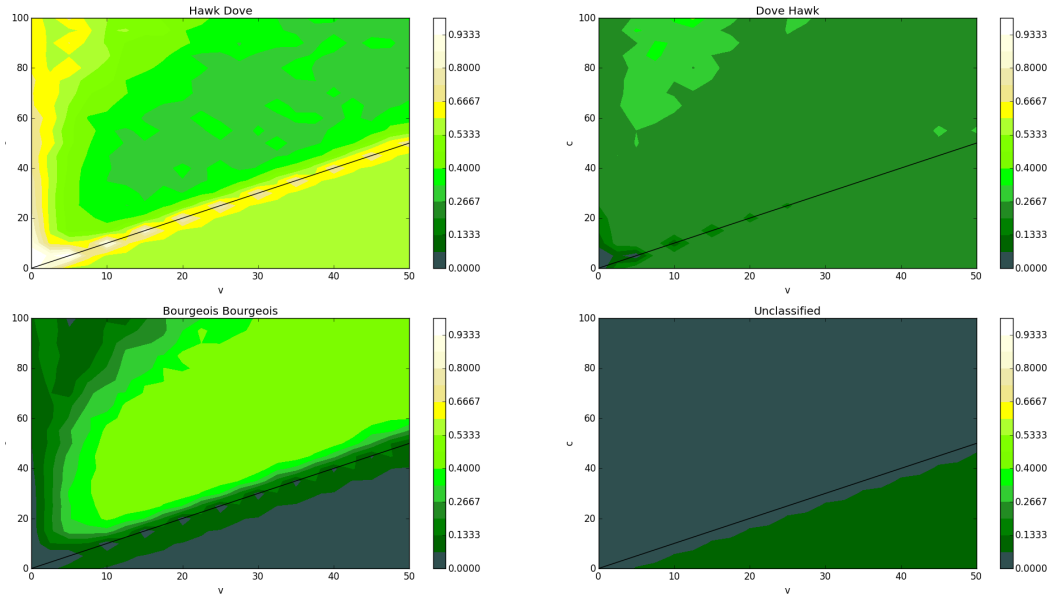

Figure 6: Contour Graph: Frequency of Each Equilibrium as a Function of the Parameters v and c

```
def test_contour_graph(self):#2d contour color plot
    s = VariedGame(HawkDoveBourgeois, WrightFisher)
    s.vary_2params('v', (0, 50, 20), 'c', (0, 100, 20), num_iterations=50, \
    num_gens=500, graph=dict(type='contour', lineArray=[(0, 50, 0, 50)]))
```

Note that the Bourgeois Bourgeois equilibrium only emerges a significant proportion of the time when $v < c$, thus validating the Nash Equilibrium analysis.

5.2 Wire Frame

Next we vary two parameters of the costly signaling game to demonstrate the wire frame graph. Here we vary lCost, the cost low senders pay to send a signal on the y-axis, and lProp, the proportion of low senders on the x-axis. The z-axis shows the frequency of the equilibrium in question, with each plot focusing on a different equilibrium.

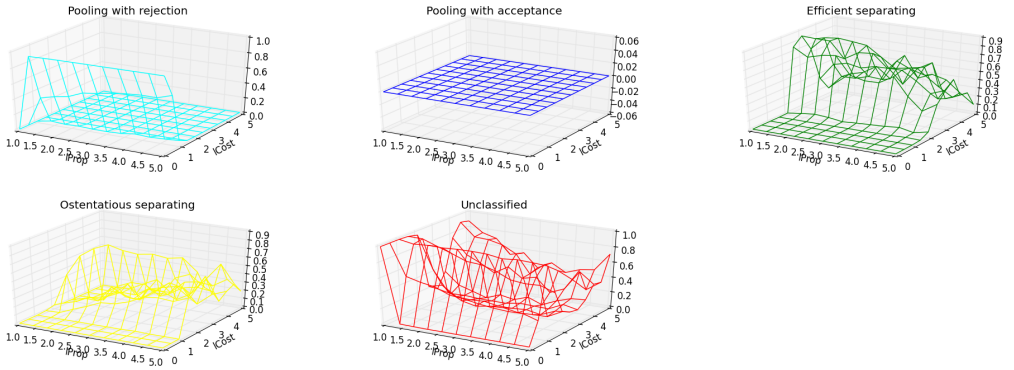

Figure 7: 3D Wire Frame Plots: Frequency of Each Equilibrium as a Function of the two cost parameters $lCost$ and $lProp$

References

DeScioli, Peter and Bart Wilson, "The Territorial Foundations of Human Property," 2011, pp. 297–304.

Hoffman, Moshe and Erez Yoeli, "Costly Signaling," University Lecture 2015.

____ , ____ , and Carlos David Navarrete, *Game Theory and Morality*, Springer International Publishing, 2016.

Hoffman, Moshe, Erez Yoeli, and Martin Nowak, "Cooperate Without Looking," *PNAS*, 2014.

Hurst, Eric, Kerwin Charles, and Nick Roussanov, "Conspicuous Consumption and Race," *Quarterly Journal of Economics*, 2008.

Irons, William, "Morality, Religion, and Human Evolution," *New York: Routledge, Inc.*, 1996, pp. 375–399.

Maynard-Smith, G. R. Price, "The Logic of Animal Conflict," *Nature*, 1973, pp. 15–18.

Nowak, Martin, *Evolutionary Dynamics*, Harvard University Press, 2006.

Spence, Michael, "Job market signaling," *The quarterly journal of Economics*, 1973, pp. 355–374.

Zahavi, Amotz, "Mate Selection - A Selection for a Handicap," *Journal of Theoretical Biology*, 1975, pp. 205–214.