Simulating Language
Feedback on assignment 1

Kenny Smith

kenny.smith@ed.ac.uk

Do we need another catch-up lab?

- We can do one of the following three options
- Two more lectures (on new stuff)
- Two more lectures, and a catch-up lab 3-5 on Thursday for those that need it
- A full 2-5pm catch-up lab Thursday, a final lecture Friday
    - What will the final lecture be about?

Quick point on Assignment 2

- **“Do we need to provide our code”?**
- If you are using code I provided, just give the filename.
- If you are doing substantial coding, I would like to see it.
- Ideally: put it online somewhere reasonably private (e.g. dropbox), include a URL in your report so I can download it if I want to.
- It must be anonymous - e.g. don’t put it at www.yourname.com
- Otherwise: paste it in to the end of your report as an appendix, doesn’t count towards the word limit

Feedback on Assignment 1: general points

- **Look at the written feedback!**
- I was pleasantly surprised by the quality on technical questions
- Less so on the longer written answers
- Top UG mark: 82
- Top PG mark: 84
- PGs: typically better on coding questions
- UGs: noticeably better on Q8

Feedback on Assignment 1: general points

- My two main tips for Assignment 2
  - Don’t waffle
  - Include graphs

Q1

1. The following python code is meant to remove all the zeros from a list and work out the average of the remaining numbers. It has a lot of bugs in it! Fix as many bugs as you can. Try to do it with the smallest number of edits possible!

```
def average_non_zero(input list)
non_zero=input_list
total=0
for i in len(input_list):
     total==total+i
     if i=0:
     non_zero.remove(i)
print "The original list was:" input_list
print 'The non-zero numbers in the list are:", non_zero
     print 'The average is:', total/len(input_list)
```

1 Question 1

The code was modified to the following version:

```
from copy import deepcopy


def average_non_zero(input_list):
        non_zero=deepcopy(input_list)
        total=0
        for i in input_list:
                total=total+i
                if i==0:
                        non_zero.remove(i)
    print "The original list was:", input_list
    print "The non-zero numbers in the list are:", non_zero
    print 'The average is:', total/len(non_zero)
```

zero list error

The following changes were made for the function to accomplish the described task:

1. included import command for `deepcopy`, as this is needed to use two different lists
2. added missing underscore in parameter `input_list` to function, as otherwise variable will not be defined when used
3. added colon after function declaration, as otherwise invalid syntax

Note: this code does not work (gives an error message) if the input_list is all zeros. To fix that, instead of the last line being:

print 'The average is:', total/len(non_zero)

you should have:

```
if len(non_zero)==0:
        print “There is no average non-zero number”
else:
        print 'The average is:', total/len(non_zero)
```

Q2

2. Briefly state how the number of iterations of a monte carlo calculation of communicative accuracy relates to the result of that calculation. Illustrate your answer with one or more graphs of simulation results. **[word limit: 100]**

2 Question 2 (95 words)

With an increasing number of iterations of a monte carlo simulation the calculation of communicative accuracy is more reliable. This is because communicative accuracy represents the probability of success in a (random) communicative event between members of a population. This probability can be measured via relative frequency and as the number of iterations approaches infinity, the relative frequency approaches the true probability. (Koehn, 2010) Such a convergence (from around 200 iterations onwards) is illustrated in Figure 1 for the communicative accuracy in Question 3.

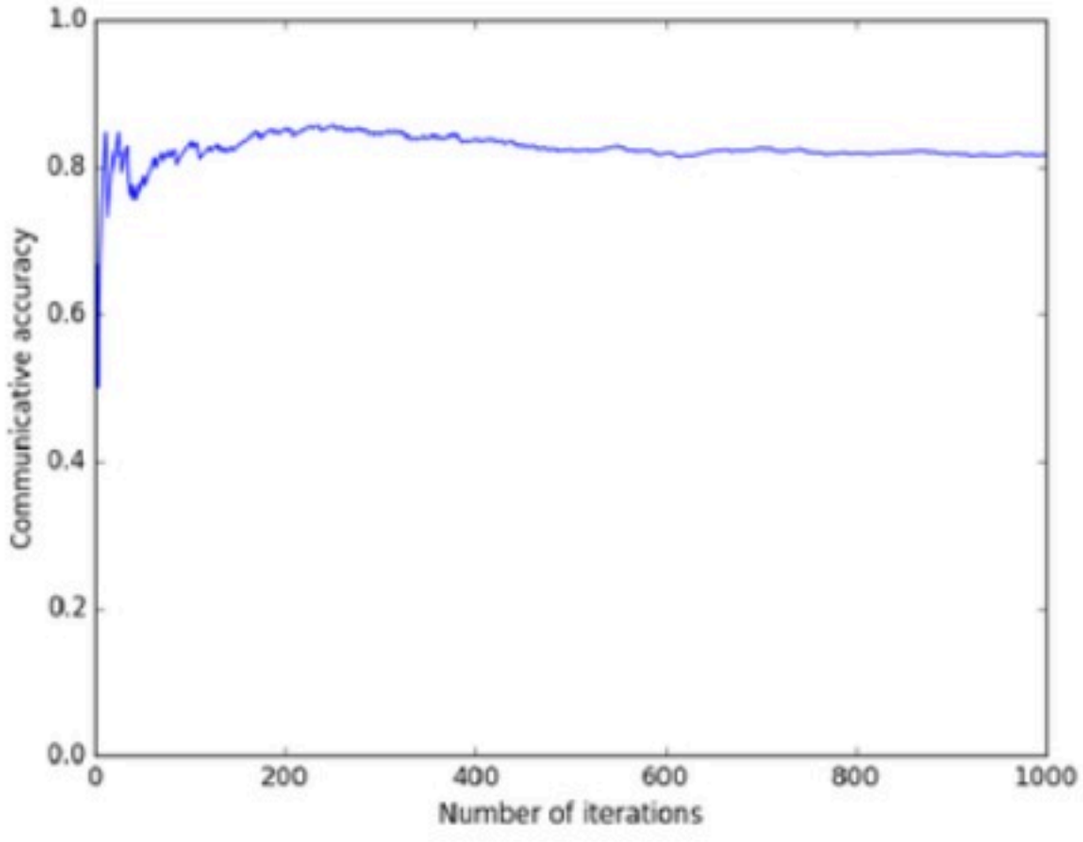


Figure 1: Dependency of communicative accuracy calculation on number of iterations

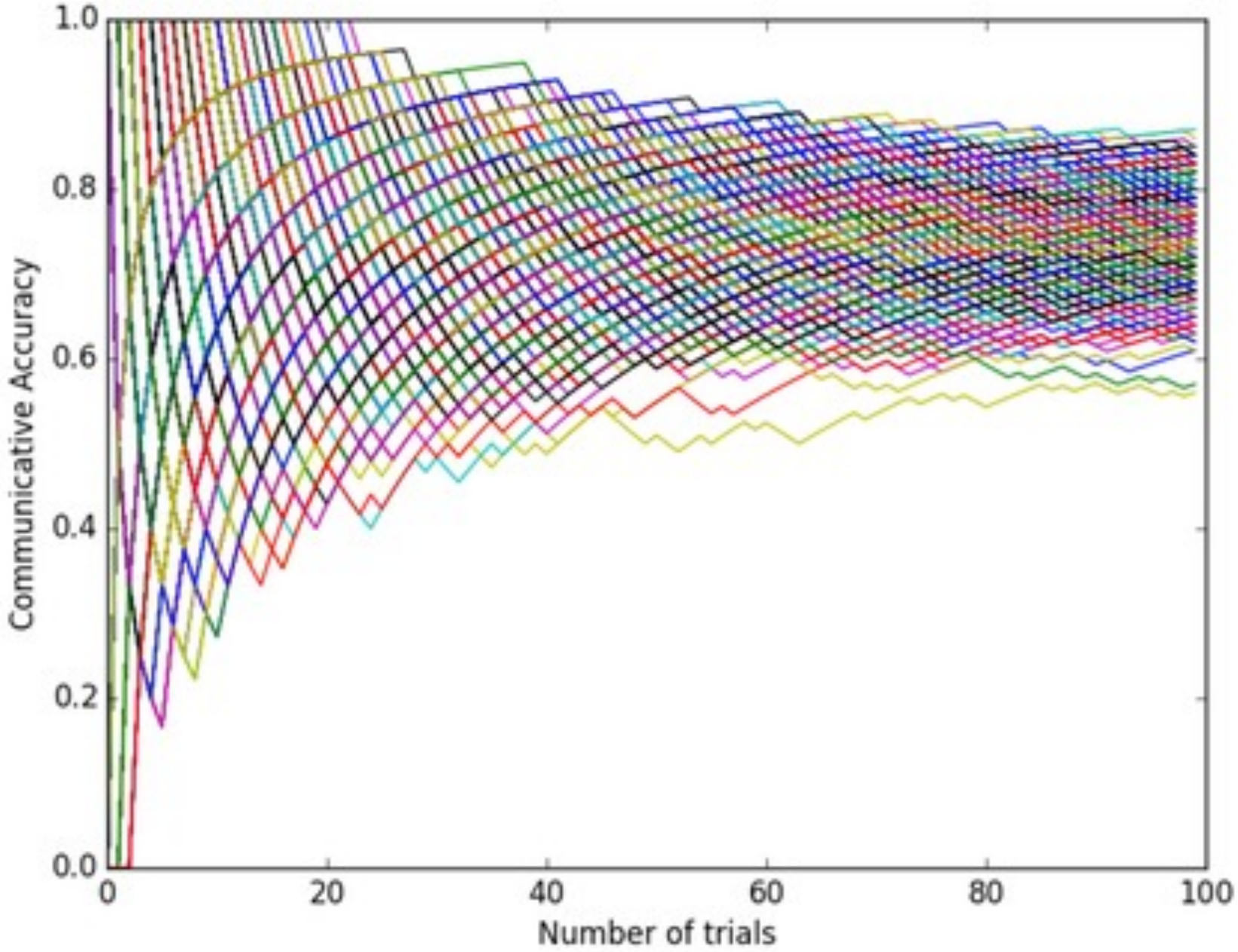
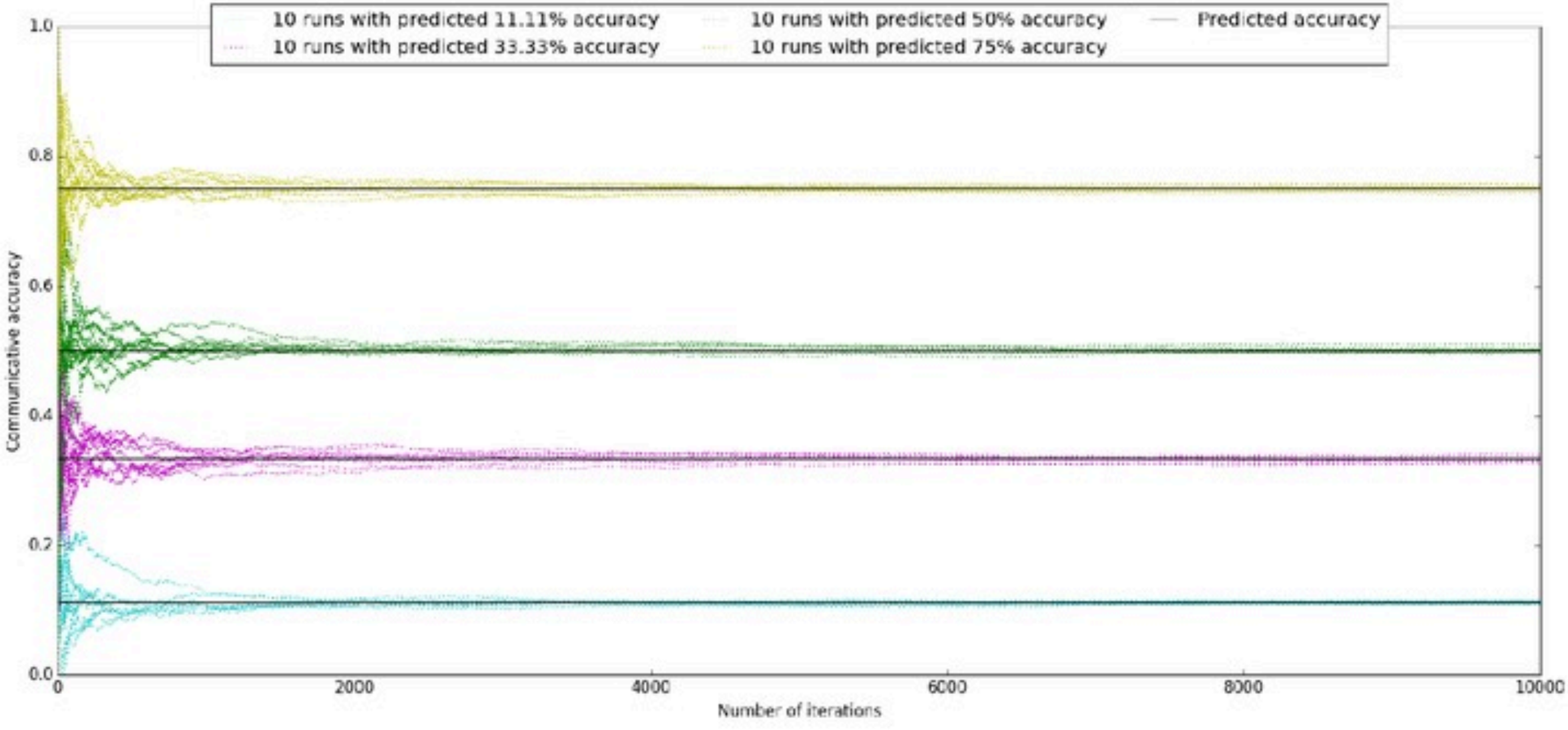

1.0
0.8
0.6
0.4
0.2
0.0
Communicative Accuracy
0
20
40
60
80
100
Number of trials

Fig. 1: Experimental and predicted accuracy over number of iterations (for 10 runs each of 4 different agent pairs)



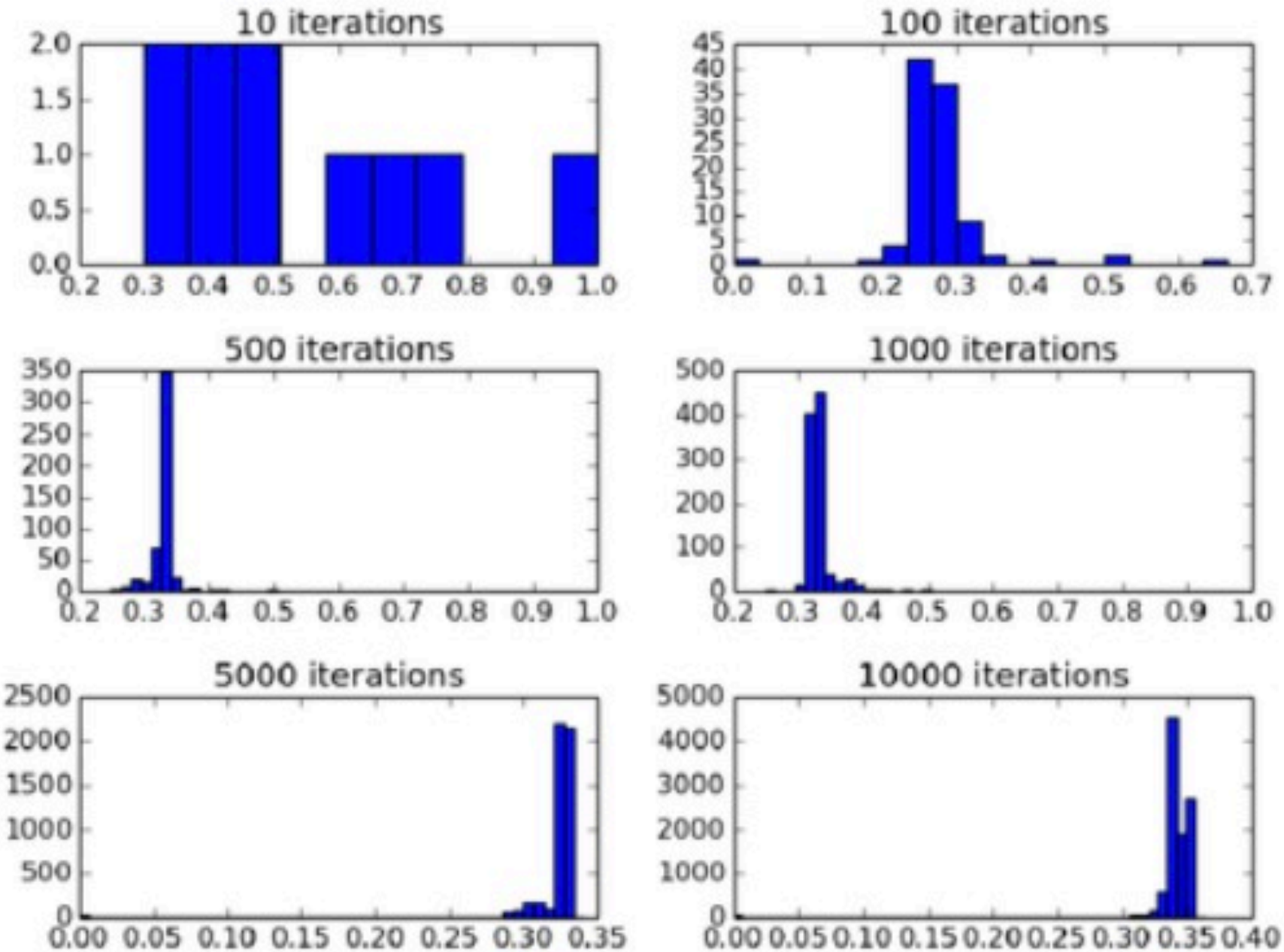


Figure 1: The results of several Monte Carlo simulations

Q3

3. Two agents with innate signalling systems are illustrated below. Assuming they use winner take all production and reception, what would be the expected communicative accuracy of a population of agents made up of 10% of type A, and 90% of type B? (i.e. what's the chance that any random communicative episode between two members of such a population will be successful?)

A:

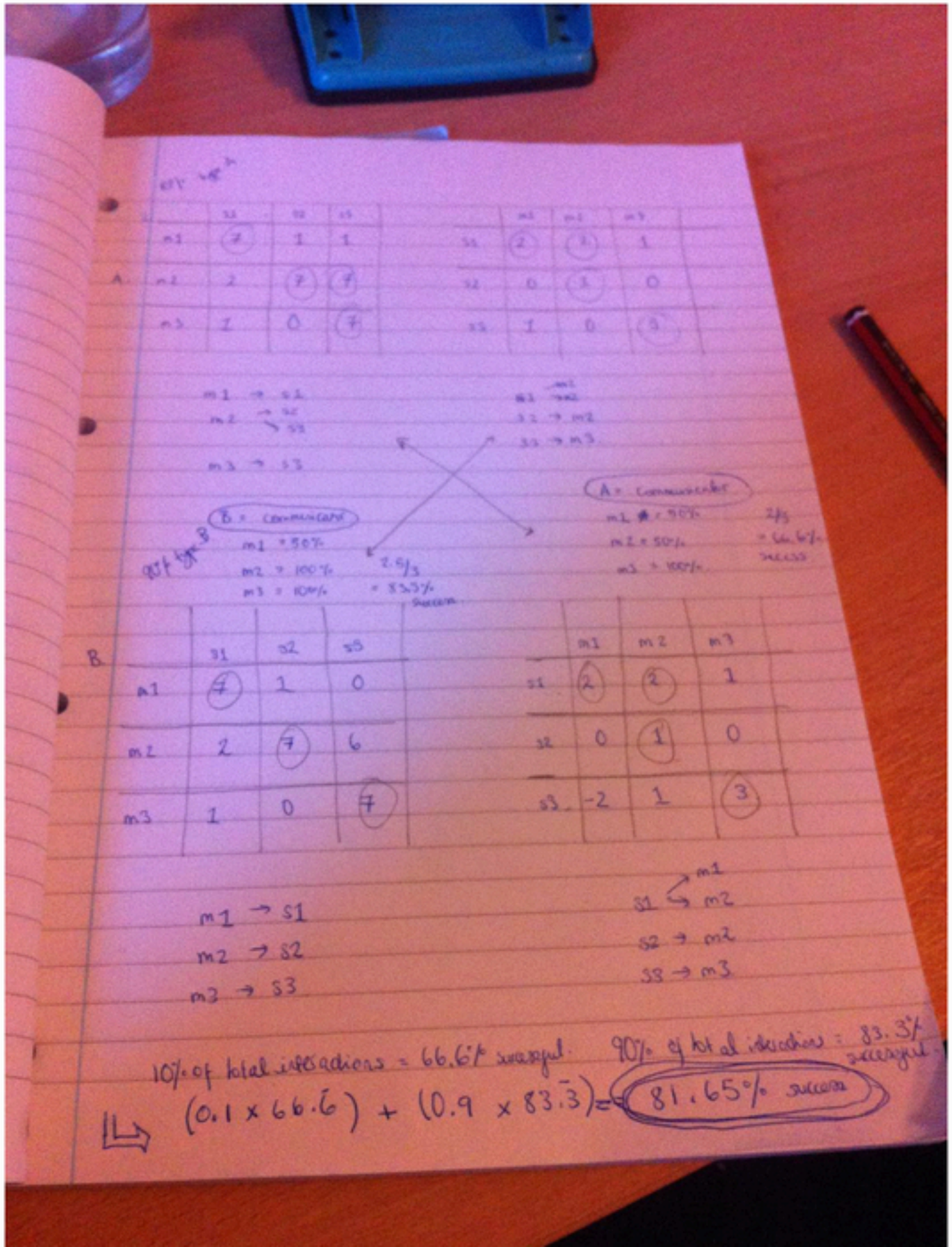
	s1	s2	s3
m1	7	1	1
m2	2	7	7
m3	1	0	7

	m1	m2	m3
s1	2	2	1
s2	0	1	0
s3	1	0	3

B:

	s1	s2	s3
m1	7	1	0
m2	2	7	6
m3	1	0	7

	m1	m2	m3
s1	2	2	1
s2	0	1	0
s3	-2	1	3


m1 → s1
m2 → s2
m3 → s3
s1 → m1
s2 → m2
s3 → m3
B = communicative
m1 = 50%
m2 = 100%
m3 = 100%
2.5/3 = 83.3% success
A = communicative
m1 = 50%
m2 = 50%
m3 = 100%
2/3 = 66.6% success
10% of total interactions = 66.6% successful
90% of total interactions = 83.3% successful
(0.1 × 66.6) + (0.9 × 83.3̄) = 81.65% success

Q4

4. Why does spatial organisation lead to communication in Oliphant's (1996) simulations? **[word limit: 300]**

- Some of you got confused about issues of reciprocity, trust
- Some of you spent too long talking about his other simulations
- Some answers were quite waffly: if you only have 300 words, every sentence needs to do a job.
- Some answers were rather vague: it’s OK to use technical terms!

4.
As demonstrated in simulation 2, when there is no direct benefit or pressure to be a good communicator, the population does not converge on an optimal signaling system. Correspondingly, spatial organization leads to optimal communication because it creates pressure to have a good production system. With spatial organization, agents are more likely to communicate with agents who are nearby, and nearby agents are likely to be closely related because offspring are placed in the same area as the parent. As a result, by being a good

producer and signaling in a way that others will understand, an agent will experience a direct benefit by increasing the fitness of its own genes.

In addition, spatial organization supports a structure where communication systems will exist in groups since offspring stay within close proximity of their parents. That is, agents with good communication systems will be close together and propagate the survival of the group, whereas agents with poor communication will also be close together, and eradicate one another. As a result, good communication systems flourish, bad communication systems die out, and the population as a whole converges to a Saussurean system.

[word count: 189]

Q5

5. Change the `pop_update` function in `signalling2.py` so that it simulates a situation in which different meanings crop up in the environment with different frequencies. Show your new `pop_update` function here.

- Lots of answers: work with exactly 2 meanings, exactly 3 meanings, etc
- Best answers: work with **any** number of meanings

Question 5

The following code is an adaption of the *pop_update* function in *signalling2.py* that simulates selection of meanings weighted by frequency, where *meaning1_freq* and *meaning2_freq* can be set to any integer values (edits in red):

```
def pop_update(population, meaning1_freq, meaning2_freq):
    speaker_index = rnd.randrange(len(population))
    hearer_index = rnd.randrange(len(population) - 1)
    if hearer_index >= speaker_index: hearer_index += 1
    speaker = population[speaker_index]
    hearer = population[hearer_index]
    meaning = rnd.choice([0]*meaning1_freq + [1]*meaning2_freq)
    success = communicate(speaker[0], hearer[1], meaning)
    speaker[2][0] += success
    speaker[2][1] += 1
    hearer[2][2] += success
    hearer[2][3] += 1
```

5. Changes bolded and explained in comments:

```
import random as rnd
def pop_update(population):
    speaker_index = rnd.randrange(len(population))
    hearer_index = rnd.randrange(len(population) - 1)
    if hearer_index >= speaker_index: hearer_index += 1    # ensure speaker
                                                              #and hearer are different
    speaker = population[speaker_index]
    hearer = population[hearer_index]
    from copy import deepcopy
    selections=deepcopy(range(len(speaker[0])))
for i in range(len(speaker[0])):
    for n in range(i):
        selections.append(i)
meaning = rnd.choice(selections)  #changes frequency choice such that later meanings are
                                  #more and more likely to be chosen (highly skewed)
success = communicate(speaker[0], hearer[1], meaning)
speaker[2][0] += success
speaker[2][1] += 1
hearer[2][2] += success
hearer[2][3] += 1
```

5. The new pop_update function below takes random meaning from a subset of the range of the meaning list.

```
def new_pop_update(population):
        speaker_index = rnd.randrange(len(population))
        hearer_index = rnd.randrange(len(population) - 1)
        if hearer_index >= speaker_index: hearer_index += 1
        speaker = population[speaker_index]
        hearer = population[hearer_index]

        p1 = rnd.randrange(len(speaker[0]))
        p2 = rnd.randrange(len(speaker[0]))
        while p1 == p2:
                p1 = rnd.randrange(len(speaker[0]))
                p2 = rnd.randrange(len(speaker[0]))
        if p1 > p2:
                start = p2
                stop = p1
        else:
                start = p1
                stop = p2
meaning = rnd.randrange(start, stop)

success = communicate(speaker[0], hearer[1], meaning)
speaker[2][0] += success
speaker[2][1] += 1
hearer[2][2] += success
hearer[2][3] += 1
```

Q6

6. The mutation function in `evolution1.py` can completely change the value of a cell in the matrix of an offspring. It might be more realistic for mutation to be gradual instead - changing a cell only slightly. Modify the code so that this happens instead. Show your new mutation function here.

- Frequently over-thought!

6. New code:

```
def mutate(system):
    for row_i in range(len(system)):
        for column_i in range(len(system[0])):
            if rnd.random() < mutation_rate:
                system[row_i][column_i] = rnd.randint(system[row_i][column_i]-1, system[row_i][column_i]+1)
                if system[row_i][column_i] > mutation_max:
                    system[row_i][column_i]=mutation_max
                if system[row_i][column_i] < 0:
                    system[row_i][column_i]=0
```

Q7

7. What corresponds to the concept of innateness in the `evolution1.py` simulation? What about in the `learning2.py` simulation? **[word limit 200]**

- Prone to waffle here - particularly the PG students!

Question 7, word count: 159

What characteristics in humans or animals might be innate is very debatable, due to innateness being a slippery concept (see for example Griffiths (2002), What is innateness?). For our purposes, I will define as innate that which is given before any input (information) has an influence on the system.
In evolution1.py we worked with innately coded **genetic algorithms** or communication matrices determining communication behaviour. Even though they were randomly generated and exposed to random mutation, their initial settings were fixed independent of any information that the system was exposed to and thus innate.
In learning2.py, on the other hand, communication weights were fixed according to specific rules. Therefore, here, an individual's **learning rule** is innate due to its creation of a bias with which the incoming information is processed. Additionally I would argue that the initial 'format' of the individual is innate, too. By this I mean the format of the '**to-be-weighted-matrix**', which determines which things can be learned.

Q8

8. When Smith (2002) talks about "learners", “maintainers” and "constructors", what does he mean? **[word limit 500]**

IMPORTANT:

For every question I expect concise answers plus, where appropriate, the use of simulation results to illustrate key points.

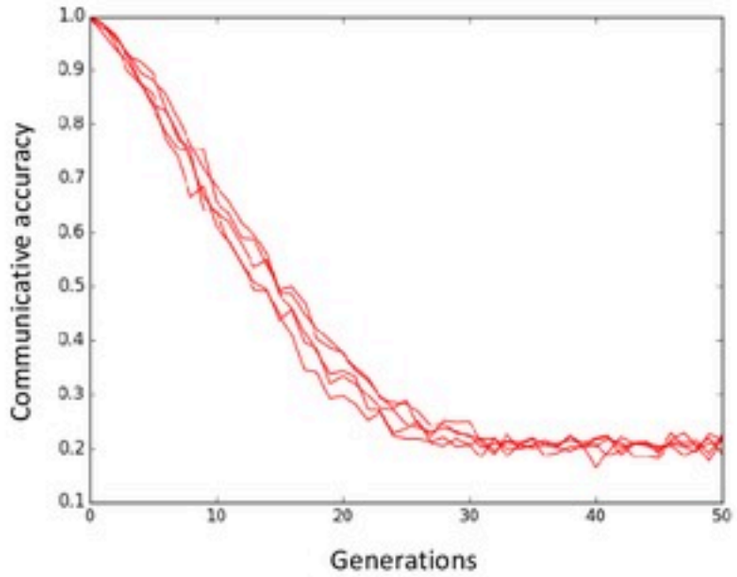
Credit will be given particularly for **clarity, brevity and precision** in writing plus **convincing use of simulations** in support of your argument.

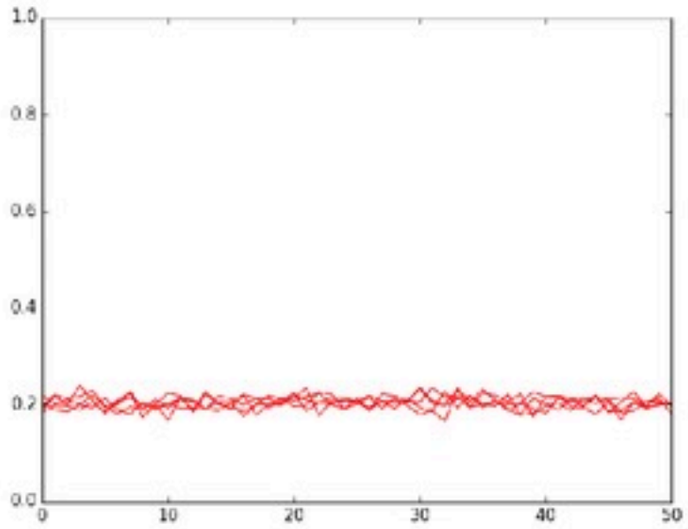
- Quite a lot of waffly answers
- Lack of precision in language used
- Lack of graphs

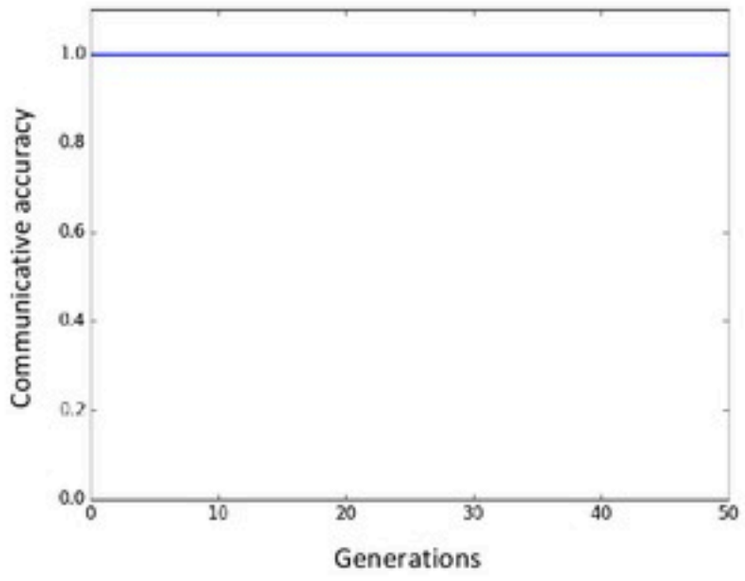
8. Smith defines “learners” as agents that have a weight update rule that allows them to acquire an optimal communication system. This was proven if the agent, after being exposed to each meaning-signal pair, always produced the correct signal when prompted by the meaning (i.e. when given m_i, the agent produces s_i) and always interpreted the signal with the correct meaning (i.e. interpreted s_i as meaning m_i). 31 of the possible 81 weight update rules allowed agents to learn. These were, simply, those rules that made stronger connections between paired meanings and signals than unpaired meanings and signals. That is, learners have update rules of the form [α, β, γ, δ], where α is the update rule for if both the meaning and the signal in question are activated, β is the rule for if the meaning is activated but the signal is not, γ is the update rule for if the meaning is not activated but the signal is, and δ is the rule for if they are both inactive, such that $\alpha + \delta > \beta + \gamma$.

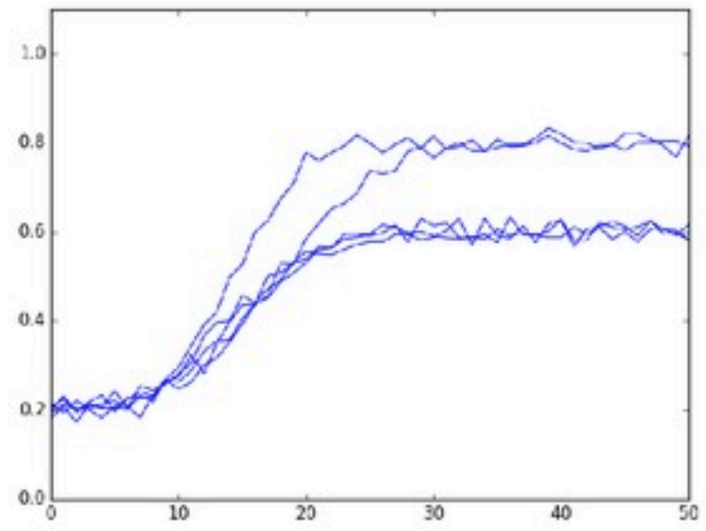
“Maintainers” are agents that, in a population, can keep an optimal communication system optimal even with a small degree of noise. This can be tested by providing the initial population with a set of optimal meaning-signal pairs, and then running a number of trials in which a random agent is removed from the system and replaced by a new agent that is trained on the communication with a set of pairs generated from the other agents in the system. Smith found that only learners can be maintainers, and only 18 of the 31 update rules that allowed learning allowed maintenance. Those update rules that allowed maintenance were such that $\alpha > \beta$ and $\delta \geq \gamma$. These rules mean that after the agent is given a meaning-signal pair, it can correctly assign the appropriate signal to that meaning, although it may not be able to generalize to ruling out the use of that signal for other meanings. This is fine in the case of maintainers because the agents are initially presented with an optimal, one-to-one communication system.

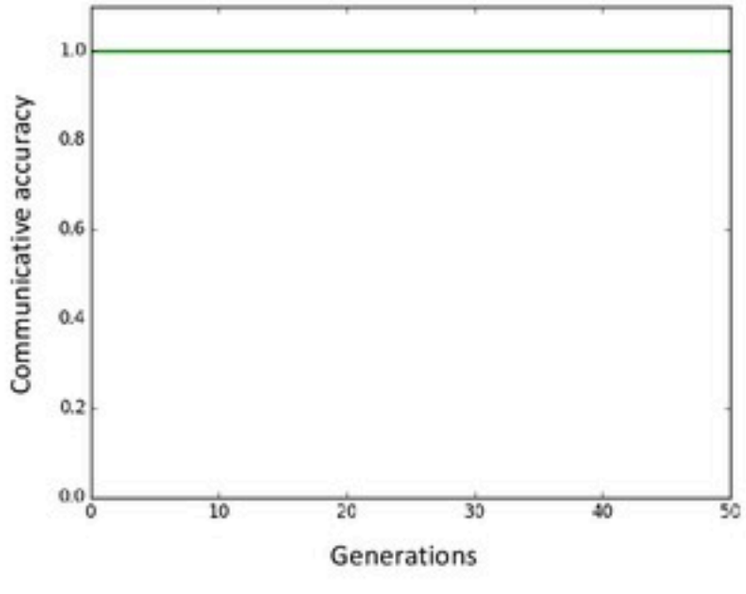
“Constructors” are agents who in a population can repeatedly create an optimal system from a random one. Constructors have weight update rules such that $\alpha > \beta$ and $\delta > \gamma$. Thus, they are able to assign the appropriate signal to its meaning, but also to generalize and to rule out using that signal for any other meaning. They are biased toward acquiring a one-to-one communication system, even from imperfect or incomplete data, which is why they are able to construct an optimal system from a random one. All constructors are also maintainers, but only 9 of the 18 maintainer rules are constructors. Below find examples of populations of learners with rule [0,1,-1,1] (red), maintainers with rule [1,0,0,0] (blue), and constructors with rule [1,0,0,1] (green) when presented with an optimal system (on the left) and a random system (on the right). (Word count: 497)

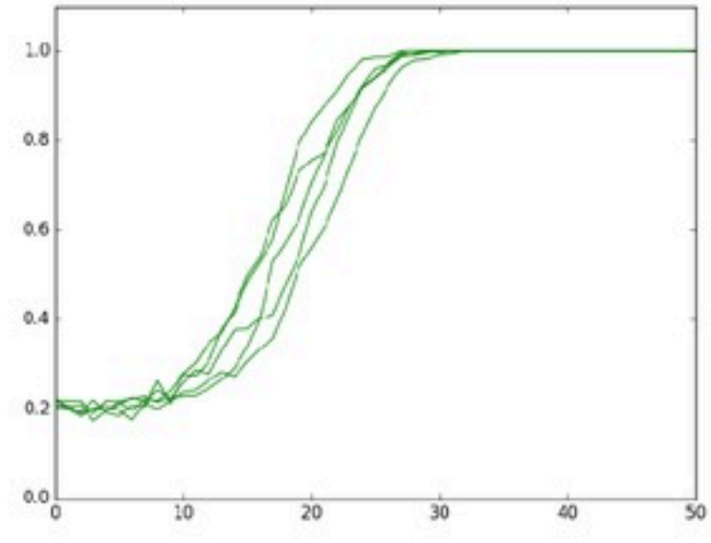

Communicative accuracy
Generations




Communicative accuracy
Generations




Communicative accuracy
Generations



My advice for Assignment 2

- **There is substantial room for improvement over what you produced for Q8**
- Be concise
  - Re-read your answer multiple times, eliminate stuff that isn't relevant or necessary
- Be precise
  - This is a technical course, I expect technical language in the write-up
- Include graphs
  - Not just random graphs - can you find a way to plot results that really captures the effect you are trying to show?
  - Make the graphs pretty - informative colours, axes labels, legends etc