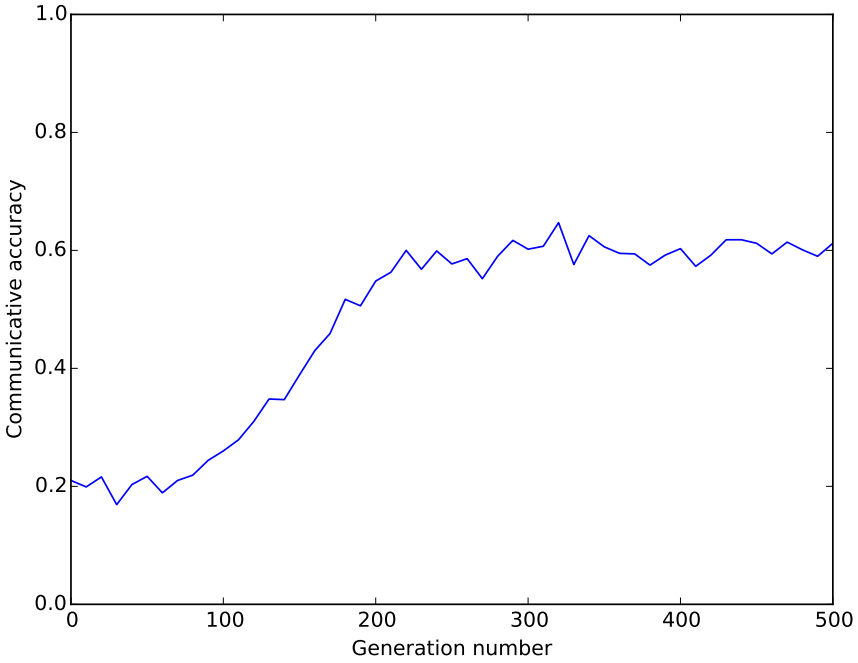
Simulating Language Lab 7

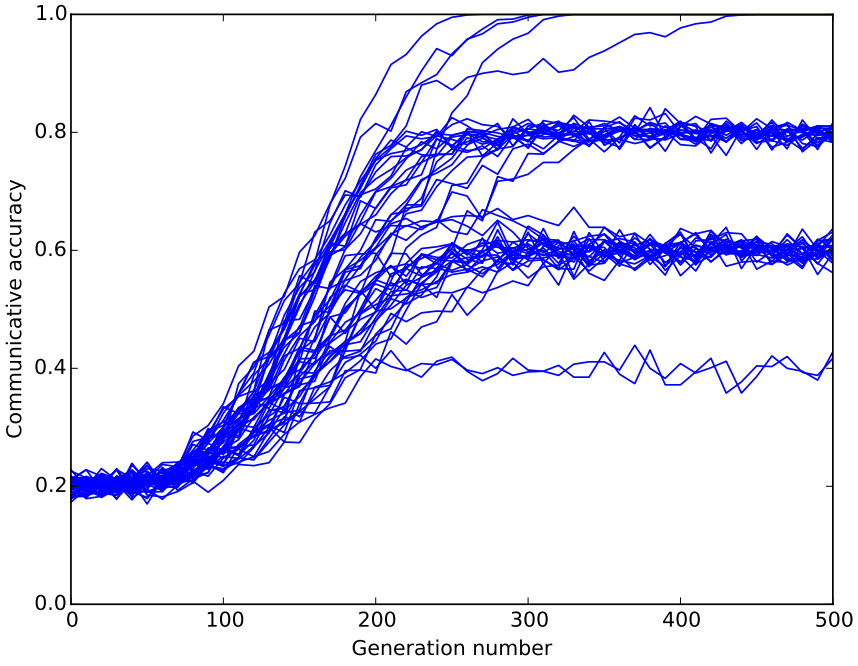
Question 1

This should be pretty straight forward. With the default parameters, you should have a graph like this:



To produce a nice graph I added axis labels, set the *y*-axis to cover the whole of the interval [0, 1] (so that all our graphs are comparable), and set the *x*-axis values to range(0, 501, 10) which gives us the numbers from 0 to 500 in increments of 10.

But of course, the graph above could be misleading. If you try running the simulation a bunch of times, you'll see that we get a wide variety of results (as you can see in the following graph).



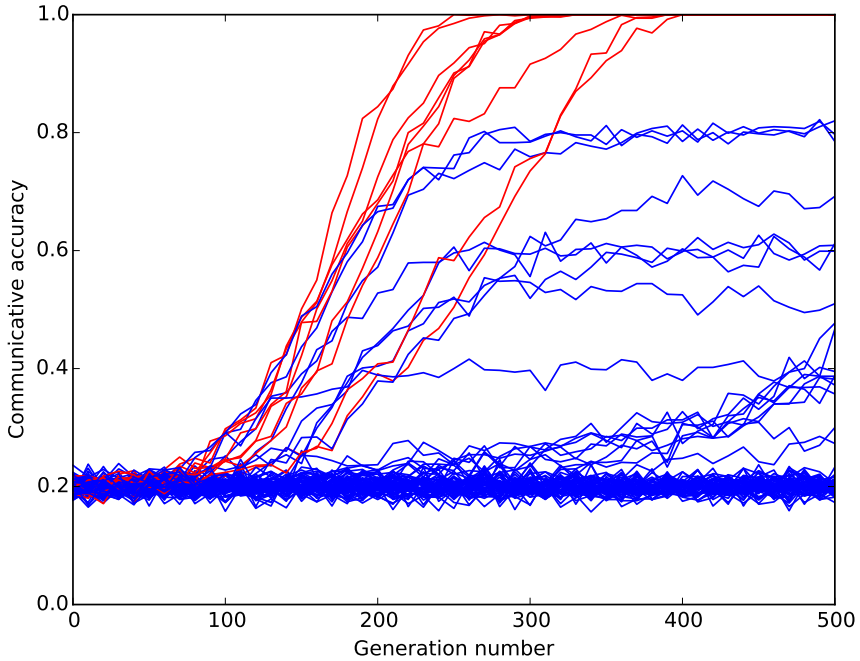
Some of the runs end up with an optimal system, but most of them don't. So in this case it's safe to assume that the default learning rule [1, 0, 0, 0] is not a constructor.

Question 2

Rather than trying rules at random, let's try all 81 possible learning rules and put them on one plot. To do this, I added an extra argument (rule) to the definition of the simulation() function and then ran the following bit of code:

```
for alpha in [-1, 0, 1]:
  for beta in [-1, 0, 1]:
    for gamma in [-1, 0, 1]:
      for delta in [-1, 0, 1]:
        pop, ca = simulation(500, 1000, 10, [alpha, beta, gamma, delta])
        if ca[-1] == 1:
          plot(range(0, 501, 10), ca, color='red')
          print [alpha, beta, gamma, delta]
        else:
          plot(range(0, 501, 10), ca, color='blue')
```

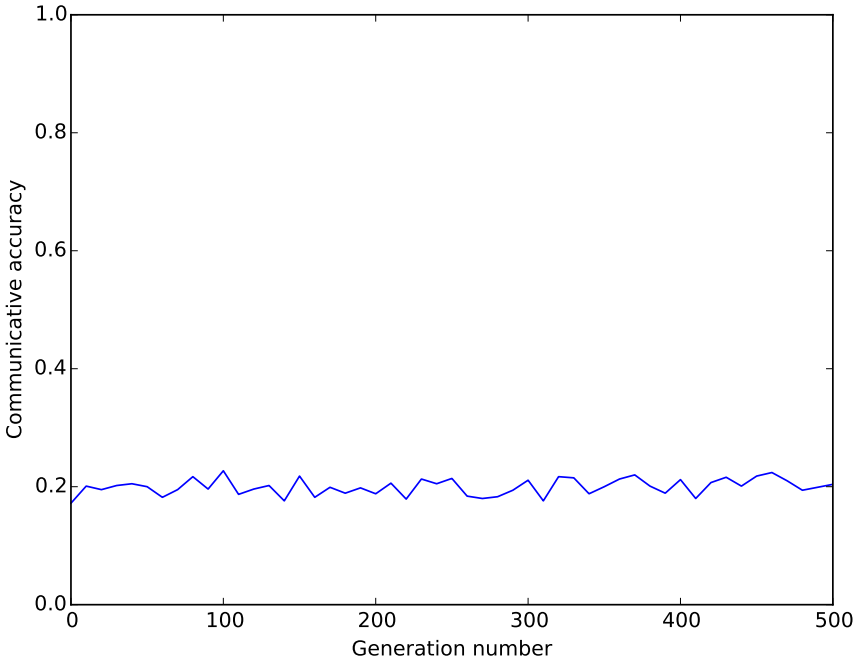
This goes through every combination of α, β, γ, and δ and runs the simulation for that rule combination. The if statement at the end checks the last ca result; if it is 1, it uses red for the plot and prints the learning rule for you (otherwise it uses the classic blue). Here's what the result looks like:



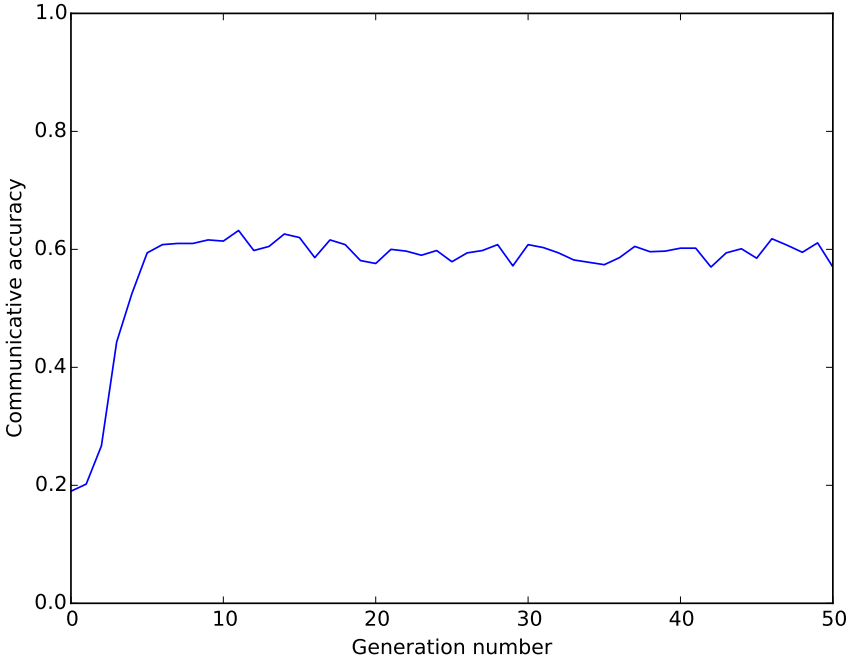
Note that 9 of the learning rules construct an optimal system (the ones in red), while the others either construct an ambiguous system or hover around chance level. (Slight problem with this: one of the non-constructor rules could by chance produce an optimal system (as we saw in Question 1) – so this approach isn't fool-proof – instead you probably want to run each of the rules a few times and average them together or something).

Questions 3

The following graph shows the default parameters with the chain method.



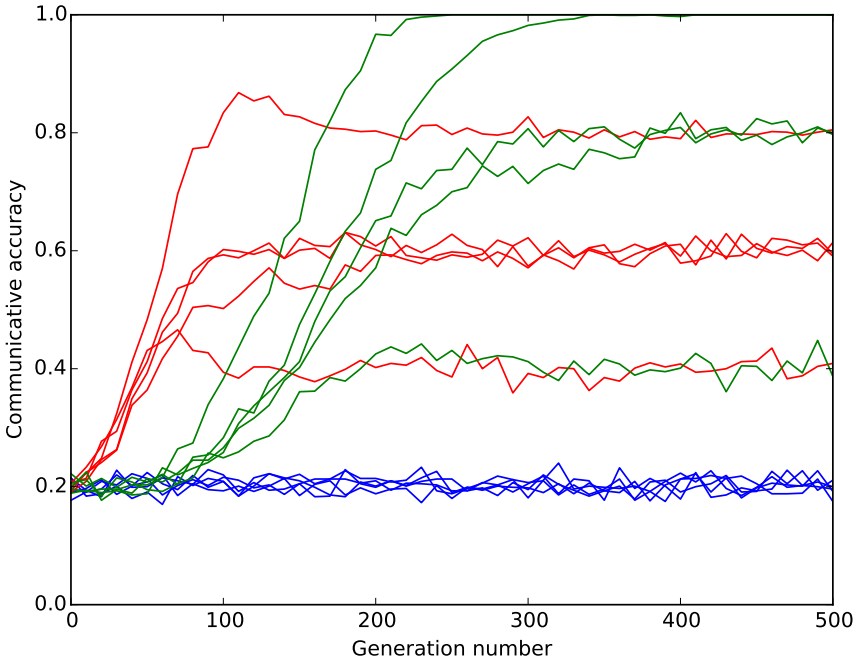
Hmmm… not much going on here. But if we do as the question suggests and increase the number of interactions by a factor of 100 and reduce the number of generations by a factor of 10, look what happens now.



It looks like the chain method can evolve some good systems, but only if you have a lot more interactions at each generation. Why do you think this is? How does the population get updated with the chain method? How does the interactions parameter influence what a new generation will look like?

Question 4

The following graph shows the chain method (blue), closed method (red), and the replacement method (green) superimposed on one plot (5 runs of each method with default parameters).



It looks like the closed method (red) evolves most quickly, the replacement method (green) evolves the strongest system, while the chain method (blue) is stuck at chance. Of course, as we saw above, the chain method seems to require more interactions – so you should take this result with a pinch of salt. Perhaps try running all three methods with more interactions and compare the results on a single plot.

Question 5

To answer this, you want to ensure the agents start with an optimal system, so first change the global parameter 'initial_language_type' to 'optimal'. You can then test out a given learning rule to see if the agents maintain the initial optimal language. What you should find is that all constructors are maintainers, but not all maintainers are constructors. In other words, constructors are a subset of maintainers. So, if a rule fails the construction test, it won't necessarily fail the maintenance test, and if a rule passes the construction test, it will definitely pass the maintenance test.

Question 6

This is a pretty conceptual question, but I think what we're looking for is some kind of setup where the language is passed on culturally while the learning rule is passed on genetically. In this way, the agents could evolve the best learning rule to learn, construct, and maintain an optimal language, while simultaneously evolving the optimal language culturally.

Spatial organization could not only determine which agents you communicate with (as we've experimented with in lab 3), but also who your 'cultural parents' are (i.e. who you learn your language from). And we could add a fitness-like factor where agents with higher communicative accuracy have more 'cultural children' (i.e. teach their language to more newborns).

Reinforcement learning could be incorporated rather straightforwardly by having the number of interactions depend on the communicative accuracy of the population at a given point in time. Or, if you wanted to make it more complex, you could evaluate the communicative accuracy for the different meanings separately and train the new agent/generation specifically on the meanings that they're not doing very well on yet.