Complexity Economics: Problem Day 5 (Group 1)

Consider the following setting:

- When people choose a technology such as a messages, the usefulness depends both on the intrinsic usefulness of the technology and its numbers of users
- The more people use a technology, the more will use it in the future
- Consider a population of agents that choose among two technologies
- In the model, people choose between two technologies sequentially, i.e. one by one
- People may consider the total number/share of users to determine usefulness...
- ...or only the choices of their neighbours
- Below you find some code chunks that might inspire you for your model (but you do not need to use them at all)

Please proceed as follows:

1. (45 min)
   (a) Discuss in the group how this system could be investigated using a python program.
   (b) Write a python program to study the problem (one python program per group).
   (c) Exchange your python program with group 2. You will be given the python program written by group 2, which deals with a different dynamical system.
2. (30 min)
   (a) Analyze and understand the python program written by group 2.
3. (15 min)
   (a) Discuss the two python programs together with group 2.

Additional notes

- Claudius and Torsten will be around. If you have any questions or if you are stuck anywhere, please feel free to ask or talk to us.
- The various code snippets listed below may be helpful in constructing the program.
- Consider commenting your code extensively. This will make it easier for the other group to understand your program.
- If you have lots of time left, try the running the simulation with different network structures:
  - Complete network
  - Multiple-ring network with size-16 neighborhoods (agents arranged in a ring, connected to the 16 nearest neighbors, 8 on either side)

  and/or with different parameters (p_i, p_d, τ_{inf}, τ_{im})

Script: Possible constructor methods of Simulation class and Agent class

```
class Simulation():
    def __init__(self):
        self.no_of_agents = 3000
        self.g = nx.barabasi_albert_graph(self.no_of_agents, 5)
        self.agents = []
        for i in range(self.g.number_of_nodes()):
            agent = Agent(self, self.g, i)
            self.agents.append(agent)
            self.g.node[i]["agent"] = agent

class Agent():
    def __init__(self, Simulation, graph, node_id):
        self.simulation = S
        self.g = graph
        self.node_id = node_id
        self.tec = None
```

Script: A possible method for the agent class to choose a method

```
    def tec_choice(self):
        # survey technology choices of the neighborhood
        tec_0_chosen = 0
        tec_total_chosen = 0
        for a in self.neighborhood:
            if a.tec in [0, 1]:
                tec_total_chosen += 1
                if a.tec == 0:
                    tec_0_chosen += 1
        r = random.uniform(0, 1)  # draw random number

        if tec_total_chosen > 0:
            if r < float(tec_0_chosen) / tec_total_chosen:
                self.tec = 0
            else:
                self.tec = 1
        else:
            if r < 0.5:
                self.tec = 0
            else:
                self.tec = 1
```

Script: Possibility for time iteration (e.g. in a run method of the Simulation class)

```
        for a in self.agentlist:
            s_total += 1
            a.tec_choice()
            if a.tec == 0:
                tec_0_chosen_total += 1
            time_list.append(s_total)
            share0_list.append(float(tec_0_chosen_total) / tec_chosen_total)
```

Script: Possibility for a method for collecting an agent's neighbors' ID numbers for the Agent class

```
    def get_neighbors(self):
        return nx.neighbors(self.g, self.node_id)
        # returns a list of agent ID numbers. Each agent can be accessed by:
```

```
            # graph_variable[agent_id]["agent"]
```

Script: Possibility for creating; running; and plotting the simulation (using the Simulation class and methods above)

```
S = Simulation()
S.run()
S.plot()
```

Script: Network generating commands for complete graphs; multiple-ring network structures; and preferential attachment networks

```
g0 = nx.complete_graph()
g1 = nx.watts_strogatz_graph()
g2 = nx.barabasi_albert_graph()
```