Complexity Economics: Problem Set Lab 5 (Group 4)

Consider the following setting:

- When people choose a technology such as a messenger, 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.
- A population of agents choose among two technologies. They choose 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. (120 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 3. You will be given the python program written by group 3, which deals with a different dynamical system.
2. (40 min)
   (a) Analyze and understand the python program written by group 3.
3. (20 min)
   (a) Discuss the two python programs together with group 3.

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 = []
        self.number_tec = 2
        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, S, graph, node_id):
        self.simulation = S
        self.g = graph
        self.node_id = node_id
        self.tec = None
        self.adoption_intention = None
```

Script: Possibility for technology choice method (in this case a method in the Agent class - it requires another method called choice_from_usage_numbers)

```
    def tec_choice(self):
        # survey technology choices
        tech_in_neighborhood = np.zeros(self.simulation.number_tec)
        for a in self.neighborhood:
            if a.tec is not None:
                tech_in_neighborhood[a.tec] += 1
        adoption_choice = self.choice_from_usage_numbers(tech_in_neighborhood)
        self.adoption_intention = adoption_choice
```

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

```
        # time iteration
        for t in range(self.max_t):
            for agent in self.agents:
                agent.tec_choice()
            for agent in self.agents:
                agent.finalize_technology_adoption()
            self.collect_statistics()
```

Script: Possible plotting method for the Simulation class (requires a 2-dimensional array recording the adoption history)

```
    def plot(self):
        """function for plotting simulation results"""
        # initialize matplotlib figure
        plt.figure()
        # set title and axis labels
        plt.title("Usage by technology")
        plt.xlabel("Time")
        plt.ylabel("Number of agents")
        # define plots
        colors = ["r", "b", "m", "g", "c", "k", "r", "b", "m", "g", "c", "k", \
                                        "r", "b", "m", "g", "c", "k"]
        for i in range(self.number_tec):
          plt.plot(range(len(self.history_agents_per_tec[i])), \
                                    self.history_agents_per_tec[i], colors[i])
        # save as pdf
```

```
        plt.savefig("tech.pdf")
        # show figure
        plt.show()
```

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 acessed 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(3000)
g1 = nx.watts_strogatz_graph(3000, 16, 0)
g2 = nx.barabasi_albert_graph(3000, 5)
```