Complexity Economics: Problem Day 5 (Group 3)

Consider the following setting:

- There is a population in a (real, not online) social network, connected by a network structure that resembles a Barabási-Albert-Network with parameter $k = 5$.
- An epidemic is spreading through the population, starting with a single, randomly chosen patient 0.
- In every time period, every infected agent may infect any of her direct neighbors in the network; every neighbor (independently) is infected with probability $p_i = 10\%$.
- In every time period, every infected agent has a probability to die (and be removed from the network) of $p_d = 1\%$
- Infections always last exactly 10 time periods ($\tau_{inf} = 10$). Agents that survive infections will be immune for a further 15 time periods ($\tau_{im} = 15$) after recovering.

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 4. You will be given the python program written by group 4, which deals with a different dynamical system.
2. (30 min)
   (a) Analyze and understand the python program written by group 4.
3. (15 min)
   (a) Discuss the two python programs together with group 4.

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, S, graph, node_id):
        self.simulation = S
        self.g = graph
        self.node_id = node_id
        self.infected = False
        self.immune = False
        self.dead = False
        self.becoming_infected = False
```

Script: Possibility for seeding the epidemic (in this case in a run function in the Simulation class)

```
    def run(self):
        # seed epidemic
        patient_zero_id = np.random.choice(self.g.nodes())
        patient_zero = self.g.node[patient_zero_id]["agent"]
        patient_zero.become_infected()
```

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.iterate()
```

Script: Possibility for collecting statistics at runtime (requires boolean variables self.immune; self.infected; and self.dead to exist (and be up-to-date) in the Agent class)

```
                self.timelist.append(t)
                immune = [a for a in self.agents if a.immune]
                infected = [a for a in self.agents if a.infected]
                dead = [a for a in self.agents if a.dead]
                self.vulnerable_list.append(self.no_of_agents - len(immune) \
                                    - len(infected) - len(dead))
                self.immune_list.append(len(immune))
                self.infected_list.append(len(infected))
                self.dead_list.append(len(dead))
                print(self.timelist[-1], self.vulnerable_list[-1],           \
                          self.infected_list[-1], self.immune_list[-1],\
                                                    self.dead_list[-1],)
```

Script: Possible plotting method for the Simulation class

```
    def plot(self):
        """function for plotting simulation results"""
        # initialize matplotlib figure
```

```
        plt.figure()
        # set title and axis labels
        plt.title("Non-infected, infected, immune, and dead agents")
        plt.xlabel("Time")
        plt.ylabel("Number of agents")
        # define plots
        plt.plot(self.timelist, self.vulnerable_list, 'g')
        plt.plot(self.timelist, self.dead_list, 'r')
        plt.plot(self.timelist, self.infected_list, 'm')
        plt.plot(self.timelist, self.immune_list, 'b')
        # save as pdf
        plt.savefig("sir.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)
```

Script: Code for removing a specific node from the network

```
self.g.node[n_id]["agent"].become_infected()
```