

## Complexity Economics: Problem Day 5 (Group 1)

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 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 ( $p_i, p_d, \tau_{inf}, \tau_{im}$ )

Script: Possible constructor methods of Simulation class and Agent class

```
1 class Simulation():
2     def __init__(self):
3         self.no_of_agents = 3000
4         self.g = nx.barabasi_albert_graph(self.no_of_agents, 5)
5         self.agents = []
6         for i in range(self.g.number_of_nodes()):
7             agent = Agent(self, self.g, i)
8             self.agents.append(agent)
9             self.g.node[i]["agent"] = agent
10
11 class Agent():
12     def __init__(self, S, graph, node_id):
13         self.simulation = S
14         self.g = graph
15         self.node_id = node_id
16         self.infected = False
17         self.immune = False
18         self.dead = False
19         self.becoming_infected = False
```

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

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

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

```
1         # time iteration
2         for t in range(self.max_t):
3             for agent in self.agents:
4                 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)

```
1         self.timelist.append(t)
2         immune = [a for a in self.agents if a.immune]
3         infected = [a for a in self.agents if a.infected]
4         dead = [a for a in self.agents if a.dead]
5         self.vulnerable_list.append(self.no_of_agents - len(immune) \
6                                     - len(infected) - len(dead))
7         self.immune_list.append(len(immune))
8         self.infected_list.append(len(infected))
9         self.dead_list.append(len(dead))
10        print(self.timelist[-1], self.vulnerable_list[-1], \
11              self.infected_list[-1], self.immune_list[-1], \
12              self.dead_list[-1],)
```

Script: Possible plotting method for the Simulation class

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

```

4     plt.figure()
5     # set title and axis labels
6     plt.title("Non-infected, _infected, _immune, _and _dead _agents")
7     plt.xlabel("Time")
8     plt.ylabel("Number_of_agents")
9     # define plots
10    plt.plot(self.timelist, self.vulnerable_list, 'g')
11    plt.plot(self.timelist, self.dead_list, 'r')
12    plt.plot(self.timelist, self.infected_list, 'm')
13    plt.plot(self.timelist, self.immune_list, 'b')
14    # save as pdf
15    plt.savefig("sir.pdf")
16    # show figure
17    plt.show()

```

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

```

1     def get_neighbors(self):
2         return nx.neighbors(self.g, self.node_id)
3         #returns a list of agent ID numbers. Each agent can be accessed by:
4         # graph_variable[agent_id]["agent"]

```

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

```

1 S = Simulation()
2 S.run()
3 S.plot()

```

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

```

1 g0 = nx.complete_graph(3000)
2 g1 = nx.watts_strogatz_graph(3000, 16, 0)
3 g2 = nx.barabasi_albert_graph(3000, 5)

```

Script: Code for removing a specific node from the network

```

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

```