



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems with MATLAB

Project Report

**Opinion Formation in a Continuous Network-based Model
and Influence of Different Network Structures**

Janosch Hildebrand Jeremia Bär Raphael Fuchs

Zurich
December 2010

Contents

1. Introduction and Motivation	3
2. Materials and Methods	3
2.1. Description of the Model	3
2.1.1. Discrete network-based model	3
2.1.2. Continuous model	4
2.1.3. Combined model	5
2.2. Networks	5
2.3. Implementation	6
2.3.1. Reproduction of the discrete model	6
2.3.2. Reproduction of the continuous model	10
2.3.3. Combined model extensions	10
2.3.4. Data generation to examine the behaviour of the model	12
3. Simulation Results and Discussion	13
3.1. Discrete network-based model	13
3.1.1. Validity of our implementation	13
3.1.2. Watts and Strogatz network	20
3.2. Continuous network-based model	24
3.2.1. Behaviour of the continuous model implementation	24
3.2.2. Simulation setup for the combined model	25
3.2.3. Group formation behaviour and interpretation of the combined model	25
4. Summary and Outlook	29
A. References	31
B. List of Figures	32
C. List of Tables	33
D. MATLAB[®] code	34
D.1. continuousModel.m	34
D.2. createRandomSocialGraph.m	36
D.3. createWattsAndStrogatzModel.m	37
D.4. generateContinuousOpinions.m	38
D.5. generateGraphs.m	38
D.6. generateOpinions.m	60
D.7. model.m	61
D.8. runContinuousModel.m	62
D.9. runModel.m	64
E. Eigenständigkeitserklärung	68
F. Agreement for free-download	69

1. Introduction and Motivation

Opinion formation is a key process in many areas of human society. Understanding the mechanisms behind opinion formation is critical for theoretical research and practical application in both politics and economics.

We studied two papers on the subject of opinion formation. In “Nonequilibrium phase transition in the coevolution of networks and opinions” [1] a network-based model with discrete opinions is presented whereas “Minorities in a Model for Opinion Formation” [2] proposes a continuous model based on random interactions of agents. We found both approaches interesting and were wondering how combining the two models would change the behaviour of the agents and influence the process of achieving consensus. Specifically, we wanted to study the effects from creating a model that uses both a network of agents, as well as continuous opinion values. Furthermore, we wanted to analyse the effect of using a more complex but theoretically more realistic initial network structure. This led us to the following research questions.

1. Considering paper [1]: How does a random graph as an approximation of social structures compare to small world social structures and what are the differences, if any? Specifically, how is the formation of opinion groups influenced when choosing this network structure over a purely random based one?
2. How does the continuous network based model¹ presented in this project compare to the discrete network model from [1] and to the continuous (not network-based) model in [2]?
 - a) Both paper [2] and paper [1] examine the achievement of consensus in their respective models. What is the influence of the network structure in the combined model compared to the combined model?
 - b) Research question 1 studies the impact of different networks structures on the opinion formation process. Can the results from the discrete model be repeated in the continuous model?
 - c) The model from [2] forms a number of disjoint opinion groups following a specific pattern. Can this pattern also be observed in our combined model?

2. Materials and Methods

2.1. Description of the Model

2.1.1. Discrete network-based model

Paper [1] tries to model opinion formation in social networks. We give a brief description of this model to allow the reader to familiarize himself with its workings.

The network is represented as an undirected graph of the N agents which represents the relationships between the individuals. The initial network is formed by randomly creating connections between a certain number of agents. The number of connections (i.e. the number of edges in the graph) is called M .

The opinion of each agent i is denoted g_i . At the beginning, these opinions are assigned randomly

¹in the following referred to as *combined model*

from a discrete set of opinions with size G . The mean number of people having a particular opinion is denoted λ , whereupon $\lambda = N/G$.

The model iteratively simulates the evolution of the network. In each step of the simulation one of two methods is chosen to model the alteration of opinions in the network. The first method simulates the termination and formation of relationships based on the individuals opinions. The second method models the propagation of opinions between agents.

To be exact, first an agent i is chosen at random. If i has no connections in the network, nothing happens. Otherwise one of the two methods is chosen.

1. With probability ϕ , terminate the connection between i and a randomly selected neighbour. Then connect i to another agent, selected at random from all agents having opinion g_i .
2. With probability $1 - \phi$, pick a random neighbour j of i and set i 's opinion g_i to j 's opinion g_j .

From now on we will refer to these as *Method 1* and *Method 2*.

2.1.2. Continuous model

Paper [2] presents a set of i independent agents holding an opinion x_i in the interval $[0, 1]$. In each iteration two randomly chosen agents meet and potentially move their opinions closer to each other. Following this mechanism, the model's attempt is not to simply represent a radical opinion change but to give account to the fact, that opinion formation is an evolutionary process taking place over time. This gradual change of opinion at the interaction of two agents is determined by the following parameters.

Opinion Threshold u : This parameter represents the fact that people who strongly disagree with each other hardly ever get closer in their opinion. That is, two agents i and j will only change their opinions if their difference is less than the opinion threshold, i.e. $|x_i - x_j| < u$.

Convergence Parameter μ : Given their social background, agents tend to change their opinions at a different pace, characterized by the parameter $\mu \in [0, 1]$.² If the difference in opinion is below the opinion threshold, the agents change their opinion according to the following rule:

$$\begin{aligned} x_i(t+1) &= x_i(t) + \mu(x_j(t) - x_i(t)) \\ x_j(t+1) &= x_j(t) + \mu(x_i(t) - x_j(t)) \end{aligned} \tag{1}$$

This model has been analysed by several research groups, as mentioned in [2]. In order to be able to reason about opinion groups, the parameter u_0 was introduced into the model. It allows the definition of opinion clusters, stating that two agents belong to the same cluster if their opinion differs by at most u_0 . The model is simulated until there is no further change in the clusterization.

The former research gives evidence that μ does not have a huge influence on achieving consensus for the whole system, but only determines the speed at which consensus is reached. Nevertheless, the research in [2] has shown that it does impact the behaviour of minorities in the system, being opinion clusters with less than 10% of the population. But that result is not of interest here. An interesting fact is the coherence of u and the number of opinion clusters. It has been shown that

²Given the formulas one can easily see that it does not make sense to choose $\mu > \frac{1}{2}$. A value of $\mu = \frac{1}{2}$ would already cause the two agents to adapt an equal opinion, i.e. to meet on middle ground. So $\mu \in [0, \frac{1}{2}]$ is assumed.

consensus can only be achieved with $u \geq 0.3$ and that the number of opinion clusters corresponds to the integer part of the formula $count(u) = \frac{1}{2u}$.

2.1.3. Combined model

In this section we present the combined model designed for our research questions. We started with the discrete network-based model and enhanced it to use continuous opinion values instead of discrete ones. This allows us to introduce the mechanisms of the continuous model into the network-based structure. As a result the model features individuals that do not simply adopt the viewpoint of a neighbour but rather shift in the direction of his opinion as in the continuous model. Furthermore, we extended the selection process of new neighbours, making it dependent on the opinion of the individuals. Any person can be selected as a new neighbour, but with decreasing, normal distributed probability corresponding to their difference in opinion. Therefore, the model introduces the new parameter `std` which is the standard derivation used in the normal distribution. In each iteration of the simulation the model randomly selects a person. If the person has no neighbours, nothing happens. Otherwise, these adjusted versions of *Method 1* and *Method 2* are used.

1. With probability ϕ , select at random one of the edges attached to the person and move the other end of that edge to a another person. The new neighbour is chosen with normally distributed probability. The expected value is the person's opinion while the standard derivation is specified by the parameter `std`.
2. With probability $1 - \phi$, the two agents interact in the same way as in the continuous model. They change their opinion according to formula (1) if their opinion differs by less than the opinion threshold μ .

2.2. Networks

To examine the impact the initial network itself has on the model, we used two different networks for comparison.

The first is a simple random graph which is also known as an Erdős-Rényi graph [3]. Random graphs are a simple and powerful way to model a social network but they lack local clustering. As a second model we implemented the so called Watts and Strogatz model to generate a graph which addresses the lack of local clustering and therefore is a better approximation of the small world structure. Those graphs are generated in the following way.

Given the number of nodes $n \in \mathbb{N}$ and the mean degree $k \in \mathbb{N}^3$ as well as a special parameter $\beta \in [0, 1]$, the model constructs an undirected graph. We assume that the nodes are labeled a_0, a_1, \dots, a_{n-1} . We first construct a regular ring lattice, a graph with n nodes, where each node is connected to k neighbours, $\frac{k}{2}$ on each side. Therefore there is an edge (a_i, a_j) if and only if $|i - j| = p$ or $|i - j| = n - p$ for some $p \in \{1, \dots, \frac{k}{2}\}$. In a second step we take for every node $a_i, i \in \{0, \dots, n - 1\}$ every edge (a_i, a_j) with $i < j$ and rewire it with probability β . That is, we replace (a_i, a_j) with (a_i, a_l) where l is chosen with uniform probability from all possible values. The possible values are all values that avoid loops ($i \neq l$) and link duplication (there is no edge (a_i, a_p) with $p = l$).

For $\beta = 0$ we get a regular ring lattice and for $\beta = 1$ a random graph. Choosing $\beta \in (0, 1)$ we interpolate between the regular ring lattice and the random graph [4]. Figure 1 visualize the Watts and Strogatz model for three different values of β . In each graph there are 20 nodes

³ k is assumed to be even

($n = 20$) and each node has four neighbours ($k = 4$). The graph on the left uses $\beta = 0$, which represents the regular ring lattice where each node has two neighbours on each side and therefore there is a strong local clustering. In contrast the graph on the right uses $\beta = 1$ and is completely random. The graph in the middle uses a value of $\beta = 0.04$, which results in a regular structure, which is modified to hold some random links. This middle graph represents a small world structure. A small world structure is a structure which has high local clustering. Additionally in such a structure each pair of nodes are connected by a short link compared to the size of the network.

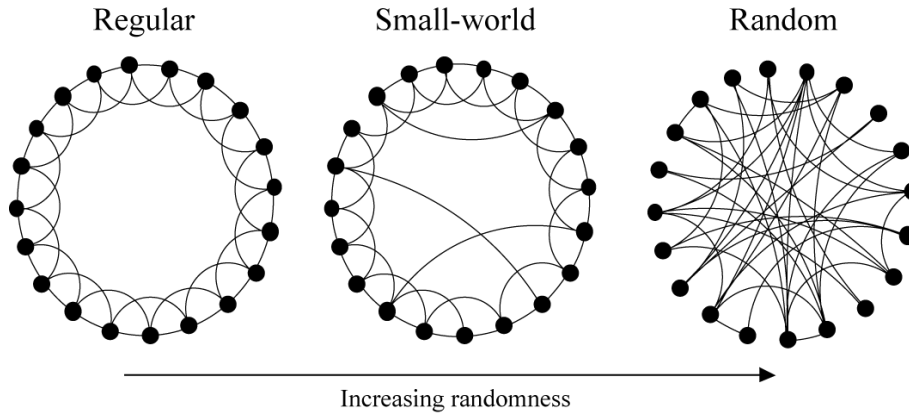


Figure 1: Different network structures for $n = 20$ and $k = 4$.

Source: http://www.cmt.phys.kyushu-u.ac.jp/kenkyu_syokai_en/NeuralNetworks/images/wsnetwork.png

2.3. Implementation

We implemented our model using MathWorks MATLAB[®]⁴. As a basis, we first reconstructed the discrete network-based model from paper [1]. Once the discrete model was functional, we enhanced it to support our desired extensions. The continuous model was implemented as a special case of the combined model. To represent the social network graph we used a binary adjacency matrix. If person i and j are connected, the adjacency matrix holds a value of 1 in row i and column j . For efficiency reasons and low memory usage we store the matrix in a sparse format. To simplify the implementation, we choose not to store the opinions in this matrix but in a separate vector instead.

For some of the simulations we used the *Parallel Computing Toolbox* of MATLAB[®] to speed up the generation of the graphs.

2.3.1. Reproduction of the discrete model

We first give a short description of all files involved in the discrete model to give a general overview and then describe the important sections of each file. In Appendix D one can find all source files in their full length.

`createRandomSocialGraph.m` This function is used to create a simple random adjacency matrix representing the relationship network. (Listing 9)

`createWattsAndStrogatzModel.m` With this function we create an adjacency matrix conforming to the Watts and Strogatz model. (Listing 10)

⁴Version 7.8 (R2009a)

`generateOpinions.m` To generate a vector of random opinions, this function can be used. (Listing 13)

`runModel.m` In the analysis we are mostly interested in opinion distributions and opinion groups distributions. We can use this function to compute those dependent on an adjustable set of parameters. The function configures the data and uses the `model` function to run the simulation and generate output data. (Listing 16)

`model.m` This function operates on an adjacency matrix representing the relationship network and an opinion vector. It is the actual implementation of the discrete model performing a certain number of simulation steps. (Listing 14)

We will now give a more detailed description of the essential parts of our implementation. Certain parts of the code are omitted for clarity.

The main loop of file `model.m` is shown in Listing 1. It simulates the interaction of the agents over a certain amount of time steps.

Listing 1: Excerpt from `model.m`

```
1 % Main loop, iterating over t
2 for t=1:iter
3
4     % Select a person and a neighbour at random
5     [...]
6
7     % Chose Method 1 or 2 with probability phi
8     if rand() <= phi % METHOD 1
9
10        % Find all people with the same opinion as person
11        opinion_group = find(opinions == opinions(person));
12
13        % Select one of the people with the same opinion at random
14        j = randi(length(opinion_group));
15        new_neighbour = opinion_group(j);
16
17        % Connect person and new_neighbour
18        % Disconnect person and old_neighbour
19        if person ~= new_neighbour
20            people(person,new_neighbour) = 1;
21            people(new_neighbour,person) = 1;
22            people(person,neighbour) = 0;
23            people(neighbour,person) = 0;
24        end
25    else % METHOD 2
26        % Adopt opinion of neighbour
27        opinions(person) = opinions(neighbour);
28    end
29 end
```

In every iteration we generate a random value (Line 8) to simulate the selection of *Method 1* (Line 8) or *Method 2* (Line 25) with probability ϕ . We then randomly choose one of the agents and a corresponding neighbour to act upon.

In *Method 1* we first select a new neighbour at random that shares the same opinion as the selected one and use it to replace the old neighbour. For this we manually adjust the adjacency matrix by connecting the person to the new neighbour (Lines 20 and 21) and breaking up the connection to the old one (Lines 22 and 23).

In *Method 2* the person's opinion is set to the one held by the selected neighbour (Line 27).

`run_model.m` enables us to run the simulation with specific parameters and to generate the opinion group size distribution which is used to analyze our implementation of the model. All the people having the same opinion form a group. The opinion group size distribution is the distribution of the number of people in those groups. The function can be configured with all the model parameters as well as to average the results over several runs of the simulation. Furthermore the model can either be run for a specific number of iterations or terminated automatically if the change of the opinion distribution over a certain number of steps falls below a specified threshold. Listing 2 shows how `run_model.m` works when run with a fixed number of iterations.

Listing 2: Excerpt from `run_model.m`

```

1  % Generate initial graph structure and opinions
2  [...]
3
4  % Simulate the requested number of steps on the model with the specified parameters
5  [~, opinions] = model(people,opinions,phi,iterations);
6
7  % Compute the opinion distribution with the histogram function 'hist'
8  % There are (n/gamma) opinions, so we want to bin the values into (n/gamma) groups
9  % We pass a vector with the exact binning points to be used
10 opinion_distribution = hist(opinions,1:(n/gamma));s
11
12 % Compute the opinion group size distribution
13 group_distribution = hist(opinion_distribution,1:n);

```

After the generation of the initial values we call the model function to run the simulation. This way we get back the opinion vector after the iterations (Line 5). Then the opinion distribution, i.e. how many people hold each opinion, is computed (Line 10). This is done by creating a histogram of the opinions. This basically counts the number of times each of the G opinions (see section 2.1.1) occurs in the final system. Finally, the opinion group size distribution is determined by creating another histogram, this time of the opinion distribution (Line 13). In this way, the number of opinion groups of same size is evaluated.

In order to average over several simulations, the mean of the opinion group size distributions of all the simulations is calculated.

Listing 3 shows how `run_model.m` operates if it is configured to determine automatically how many iterations should be performed.

Listing 3: Excerpt from `run_model.m`

```

1  % Generate initial graph structure and opinions
2  [...]
3
4  for j=1:(max_iterations/d_it)
5      % Continue to run the simulation with the specified parameters
6      [people, opinions] = model(people,opinions,phi,d_it);
7
8      % Compute the opinion distribution with the histogram function 'hist'
9      % There are (n/gamma) opinions, so we want to bin the values
10     % into (n/gamma) groups
11     % We pass a vector with the exact binning points to be used
12     opinion_distribution = hist(opinions,1:(n/gamma));
13
14     % Add the momentary opinion distribution
15     % Only computed for one computation (i.e. the last averaging step)
16     if (i==average_iterations)
17         opinion_dist(:,j) = opinion_distribution';
18     end
19

```



```

20 % Compute the opinion group size distribution
21 group_distribution = hist(opinion_distribution,1:n);
22
23
24 % Compute the normed difference between the last steps
25 dist_old = dist_new;
26 dist_new = group_distribution;
27 diff = norm(dist_new-dist_old)/norm(dist_new);
28
29 % Add the normed difference between the last steps
30 % Only computed for one computation (i.e. the last averaging step)
31 if (i==average_iterations)
32     step_differences(1,j) = j*d_it;
33     step_differences(2,j) = diff;
34 end
35
36 % Abort if threshold is reached
37 if (diff < threshold)
38     break;
39 end
40 end

```

After the data structures have been initialized the simulation is run in small, user configurable steps. This is done in a loop (Line 4) in which the simulation is first run for a small amount of iterations (Line 6). In order to be able to later continue with the simulation, the data structures need to be updated. The opinion values and the adjacency matrix representing the agent graph are both changed by running the simulation, so both need to be extracted afterwards (Line 6). The opinion distribution is then calculated (Line 12) and stored (Lines 16-18). The aggregated opinion distributions over time are optionally returned as a result of the function. This enables examination of the opinion distribution development over time.

Now the opinion group size distribution is calculated (Line 21) and compared with the result from the last simulation step (Lines 25, 26 and 27). This is done by computing the normed difference between those two steps, which can be thought of as the relative amount of change in the opinion group size distribution since the last simulation step. This data is also aggregated and optionally returned (Lines 31-34).

If this difference is smaller than the requested threshold, we exit the loop (Line 38). Otherwise, the loop begins anew and so the next simulation step is computed. For a threshold of 0 the computation will continue until the model shows no further change over the supplied iteration step size. With a negative threshold the model can be forced to run until the maximum number of iterations is reached. This is useful to accumulate all the data between steps over an exact number of iterations, even if the model does not show any further change.

To create the adjacency matrix representing the Watts and Strogatz model we use the file `createWattsAndStrogatzModel.m`. An Excerpt of it is shown in Listing 4.

Listing 4: Excerpt from `createWattsAndStrogatzModel.m`

```

1 % create sparse nxn matrix A representing regular ring lattice (step 1)
2 [...]
3 A = sparse(row_index, col_index, ones(n*k, 1), n, n, n*k);
4
5 % For every node take every edge (n_i, n_j) with i < j and rewire it
6 % with probability beta. (step 2)
7 for i=1:n
8     neighbours = find(A(i,:));
9     for j=neighbours
10         if (i < j) && (rand < beta)

```

```

11     r = i;
12     % choose until candidate found that avoid loops and link duplication
13     while ( (r == i) || (full(A(i,r)) == 1) )
14         r = randi(n); % choose random integer number
15     end
16     A(i,j) = 0; A(j,i) = 0; % reset edge
17     A(i,r) = 1; A(r,i) = 1; % new edge
18 end
19 end
20 end

```

As described in Section 2.2 we first create a regular ring lattice (Line 3). In a second step we now modify this structure. We loop over every node of the graph, which is represented as a row in the matrix, (Line 7) and replace every neighbour with probability β by a randomly chosen node, which is not yet connected to the current node. (Lines 16 and 17)

2.3.2. Reproduction of the continuous model

The continuous model can be simulated by setting $\phi = 0$ in the combined model, which results in only using *Method 2* of the combined model, and using a complete graph as the network structure.

2.3.3. Combined model extensions

The extension features the following additional files.

`generateContinuousOpinions.m` This function generates uniform distributed values as opinions for the continuous model. (Listing 11)

`continuousModel.m` This is the main file implementing the combined model and returning a histogram of it. (Listing 8)

`runContinuousModel.m` This is the overall setup function. It sets all parameters required for the model which is then run. It is responsible for processing the data generated and plotting the change in opinion over time. (Listing 15)

As this model is an extension of the network-based model, a lot of its functionality has already been explained. The important changes in `continuousModel.m` are explained in detail here.

Listing 5 introduces two important data containers used during computation. The histogram in (Line 3) is the main return value of the model. It keeps track of the change of opinion in the system over time and aims to be a basis to produce an overview similar to *Figure 2* in paper [2]. The columns represent the time scale. The model parameter `clusters` determines how many different opinion groups should be formed out of the continuous opinions existing in the system. Thus, the matrix `histogram` has (`clusters` + 1) rows leaving the last row to keep track for the norm of change since the last examination.

The combined model creates new connections between agents based on a probability in opinion difference. This requires a sorted array of opinions for an efficient computation of the opinion difference. However, each agent is uniquely identified by its array index. Sorting the array results in an inconsistency between the opinion of agents and the adjacency matrix representing their social network - especially when constantly changing the opinion during the simulation process.

The helper variable `sorted_opinions` addresses this problem. It provides a mapping between the opinion and the array index which is updated whenever opinions change in the system.

Listing 5: Data organisation in `continuousModel.m`

```

1 % histogram stores the count of poeple holding the opinion in each cluster
2 % and the norm in clusterization change.
3 histogram = zeros(clusters + 1,floor(iter/cskip));
4
5 % in order to find people with a similiar opinion, a sorted copy
6 % of the opinions is created with a link to the original person
7 % index: sorted_opinions = [ opinion , index ]
8 sorted_opinions = [opinions , (1:n)'];
9 sorted_opinions = sortrows(sorted_opinions);

```

Listing 6 shows the implementation of *Method 1* of the combined model which determines a person that shares a similar opinion. First, the opinion the new neighbour should have is determined according to the specification of the model (Lines 3-24). If the probability in (Line 4) is out of range, it retries at most 100 times to achieve a valid opinion and defaults to 0 or 1 respectively if it fails. Then in (Lines 26-46), the person with the closest opinion is selected as a new neighbour.

Listing 6: Implementation of *Method 1* in `continuousModel.m`

```

1 % METHOD 1
2 [...]
3 % figure out what opinion the new neighbour should have
4 neighbours_opinion = opinions(person) + std*randn();
5 % do not accept opinions out of range [0,1]
6 % try to find another person
7 acceptable = 0;
8 for i=1:100
9     if neighbours_opinion < 0 || neighbours_opinion > 1
10         neighbours_opinion = opinions(person) + std*randn();
11     else
12         acceptable = 1;
13         break;
14     end
15 end
16 % if that person has such an extreme opinion that it can't find
17 % anyone within range [0,1], the extrema is taken
18 if acceptable == 0
19     if neighbours_opinion < 0
20         neighbours_opinion = 0;
21     elseif neighbours_opinion > 1
22         neighbours_opinion = 1;
23     end
24 end;
25
26 % find the person with the closest opinion to
27 % neighbours_opinion
28 i = find(sorted_opinions(:,1) > neighbours_opinion, 1 , 'first');
29 if size(i,1) == 0
30     % no one has such a strong (-> 1) opinion.
31     % Take the person with the strongest opinion
32     new_neighbour = sorted_opinions(end,2);
33 elseif i == 1
34     % there is no person with a lower opinion, so the person
35     % at index i is the new neighbour
36     new_neighbour = sorted_opinions(i,2);
37 else
38     % the person at index i-1 might be closer to the opinion

```

```

39     diff_n1 = neighbours_opinion - sorted_opinions(i,1);
40     diff_n2 = neighbours_opinion - sorted_opinions(i-1,1);
41     if abs(diff_n1) < abs(diff_n2)
42         new_neighbour = sorted_opinions(i,2);
43     else
44         new_neighbour = sorted_opinions(i-1,2);
45     end
46 end

```

Listing 7 shows the implementation of *Method 2* which is a slightly modified version of the continuous model presented in paper [2]. After changing the opinion of the agents (Lines 5, 7-8), the change in opinion is also applied to the sorted list of opinions (Lines 10-14).

Listing 7: Excerpt from `continuousModel.m`

```

1  % METHOD 2
2  [...]
3
4  diff = opinions(neighbour) - opinions(person);
5  if( abs( diff ) <= u )
6      % update opinion vector
7      opinions(person) = opinions(person) + mu*diff;
8      opinions(neighbour) = opinions(neighbour) + mu*(-diff);
9      % pass update to sorted vector
10     altPerson = sorted_opinions(:,2) == person;
11     altNeighbour = sorted_opinions(:,2) == neighbour;
12     sorted_opinions(altPerson,1) = opinions(person);
13     sorted_opinions(altNeighbour,1) = opinions(neighbour);
14     sorted_opinions = sortrows( sorted_opinions );
15 end

```

2.3.4. Data generation to examine the behaviour of the model

The file `generateGraphs.m` (Listing 12) is used to run the simulations that generate the figures presented in this paper. It contains different sections for the discrete, continuous and combined model with several model configurations. The file makes use of the MATLAB[®] cell mode feature. Each section that starts with a double comment `%%` can be run as a separate execution unit.⁵ Although the file may seem rather complex, large parts are repeated several times and differ only in a few configurations. This allows for computation of different model configurations in parallel. Each simulation in itself is completely sequential as using MATLAB[®]'s *Parallel Computing Toolbox* would have required considerable restructuring of our code. Therefore the models themselves do not make use of today's multicore computer systems. Instead we used the *Parallel Computing Toolbox* when running several simulations with varying parameters. As running the file as a MATLAB[®] script takes several days if not weeks, using cell mode, we could run several cells in parallel on several multicore machines, using multiple instances of MATLAB[®]. Note that comments in this file are rather concise. Since the code is simply setting parameters, running simulations and generating plots, this should not constitute a problem.

⁵For further information on cell mode, please refer to the MATLAB[®] documentation.

3. Simulation Results and Discussion

3.1. Discrete network-based model

3.1.1. Validity of our implementation

Our first objective was to recreate the discrete model described in paper [1]. Since this formed the basis for all our later efforts, we decided it was necessary to verify our variant. To this end, we decided to try to recreate some of the results from paper [1]. We chose the first simulation as our comparison test. In this simulation a histogram of group sizes in the consensus state is generated. The consensus state describes the state in which the model has degraded into opinion groups that are no longer connected to each other but only within themselves. In this state the model will not be undergoing any further changes in terms of opinions. The results from paper [1] can be seen in *Figure 2* of the paper.

Fortunately, most of the simulation parameters used to generate these results were also indicated. The supplied parameters are shown in Table 3.

Parameter	Value	Description
ϕ	0.04, 0.458 and 0.96	Probability value for choosing <i>Method 1</i> or <i>Method 2</i> of the discrete model
N	3200	Number of agents in the model
k	4	Number of connections of each agent, i.e. number of other agents know to each one
γ	10	Opinion ratio – <i>Number of opinions</i> = N/γ
Averaging iterations	10^4	The model will be run this many times and the resulting data averaged over the intermediate results

Table 1: Parameters used in paper [1] to create *Figure 2*.

For our simulations we chose to use the same values. This allows for direct comparison with the results from *Figure 2* in paper [1]. The above table supplied us with all but one parameter required to run the simulation ourselves. In order to be able to run the simulation we needed to know the number of iterations that have to be performed in order to reach consensus state. To determine this we ran the simulation piecewise, always stopping after a certain number of iterations. We then measured the differences between the resulting data from these stopping points and tried to determine the number of iterations necessary in order to arrive in consensus state. Obviously consensus state is reached when the data does not change anymore. The actual opinion data does not change every iteration even before consensus is achieved. This is because an iteration might produce a change in the model that does not have any influence on the opinion values. So it was important to set the number of iterations between each measurement sufficiently high in order not to come to any false conclusions. Furthermore, the number of necessary iterations obviously varies between simulations. In order to easily make such measurements we ran the model for an extremely large number of iterations while frequently collecting data about the continuing differences between the iterations.

We found that the number of required iterations depends heavily on the value of ϕ and so we determined the number for each value of ϕ separately. What we found was that for $\phi = 0.96$ the model stopped changing very quickly, mostly after at most 50 000 to 60 000 iterations. Figure 4 provides an example of such a simulation. With $\phi = 0.458$ it never took more than 100 000 to 150 000 iterations to reach consensus. For an example, see Figure 3. When ϕ was set to 0.04, the required number of iterations was much higher, ranging from five million up to well over thirty-million iterations. See also Figure 2. This difference is not unexpected. As described in paper [1], for $\phi \rightarrow 1$ edges are moved to agents sharing opinions, thus disconnecting groups of different opinions. In contrast, for $\phi \rightarrow 0$ only opinions are changed and so the simulation only reaches consensus when all connected agents have equal opinions.

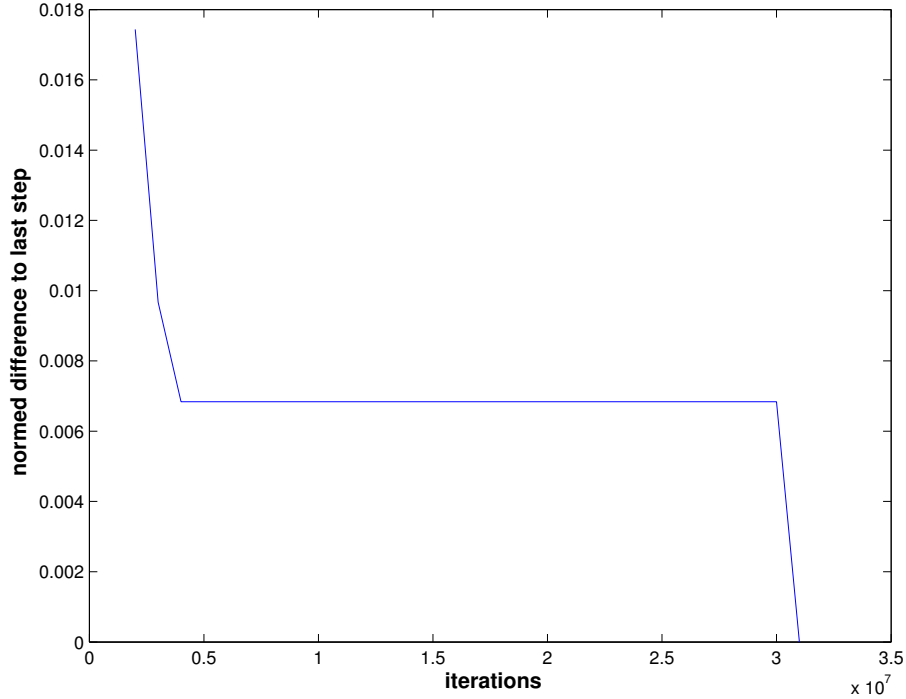


Figure 2: Convergence of model $\phi = 0.04$

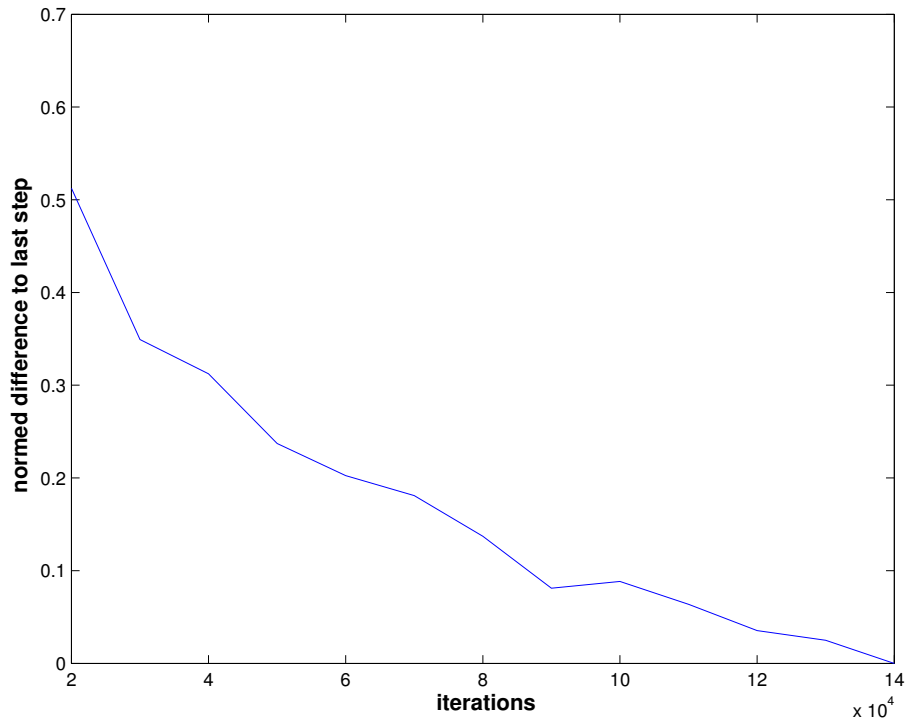


Figure 3: Convergence of model $\phi = 0.458$

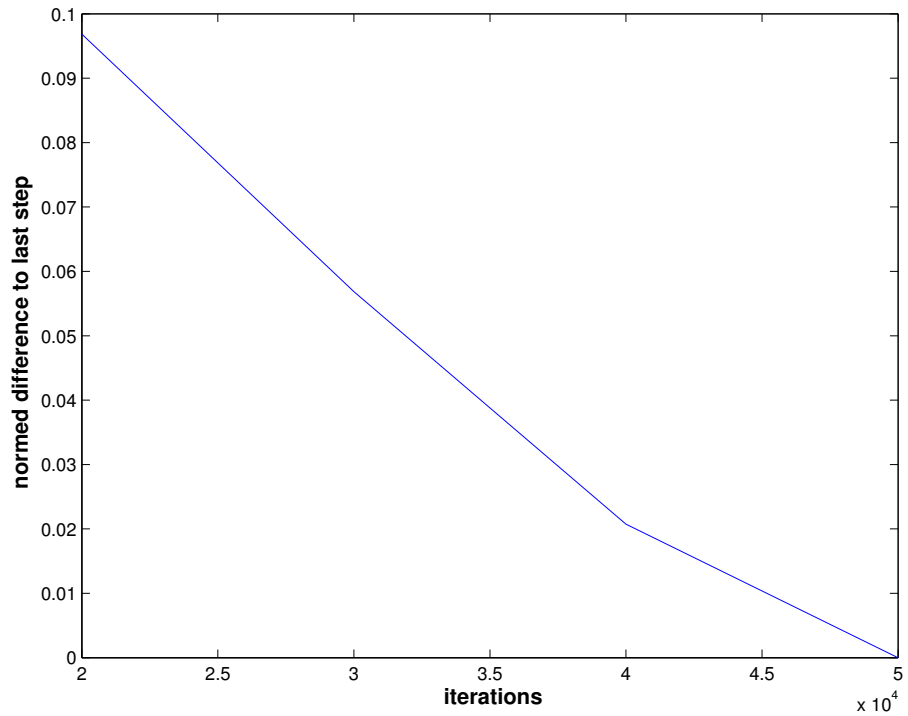


Figure 4: Convergence of model $\phi = 0.96$

We now had some rough estimates for the number of iterations necessary to run the simulations until consensus is reached. We now set out to run the simulations and to produce an equivalent to *Figure 2* from paper [1] in order to compare the models.

Regrettably we very quickly realized that with our limited computational power we would not be able to produce those results in the available time. Running a single simulation for thirty-million iterations could take up to an hour which made averaging over 10 000 simulations an impossible task. We decided to reduce the number of averaging simulations to 50. Also we used rather conservative numbers of iterations. We would have preferred to set the number of iterations well above our estimated bounds to ensure consensus for all simulations. Instead we chose to run twenty-million iterations for $\phi = 0.04$ and $\phi = 0.458$. For $\phi = 0.96$ we only ran one million iterations per simulation. With these values we expected to get at least reasonably close to consensus for small ϕ while being more than sufficient for larger values.

Before we compare the results of our model to those made in paper [1], we want to spend a moment to describe the actual observed behaviour of our model. Figures 5, 6 and 7 show the change of the opinion distributions over the indicated number of iterations for each value of ϕ for three particular simulations.

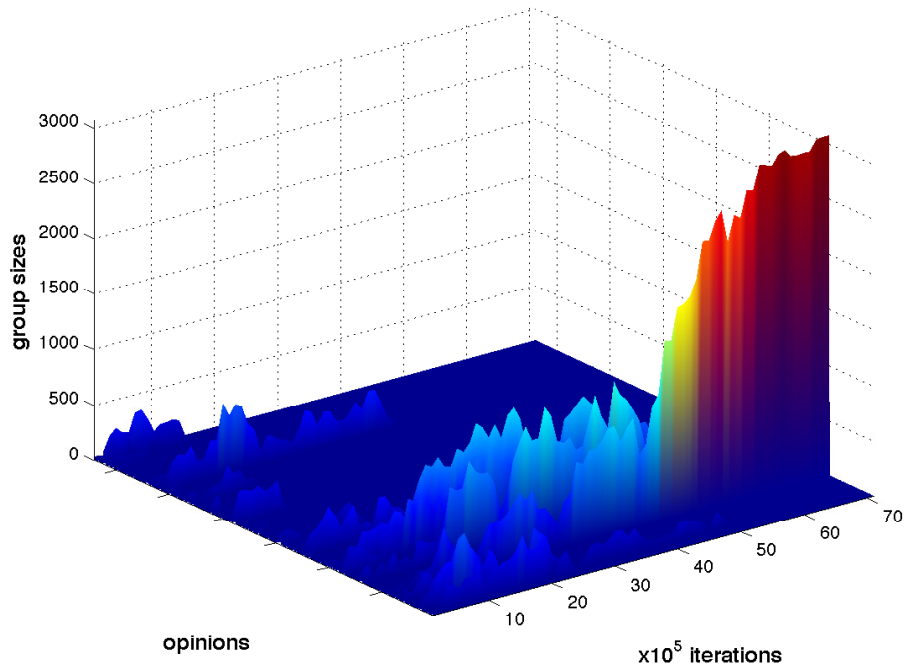


Figure 5: Opinion development over time for $\phi = 0.04$

We can see that as expected the opinions change intensively. At the beginning most opinion groups have sizes between 0 and approximately 100 with a mean of $\gamma = 10$. Very quickly several larger groups emerge. When the simulation is run long enough, eventually the largest opinion group dominates and absorbs all agents in the system it can connect to. When consensus is reached only one giant community is left.

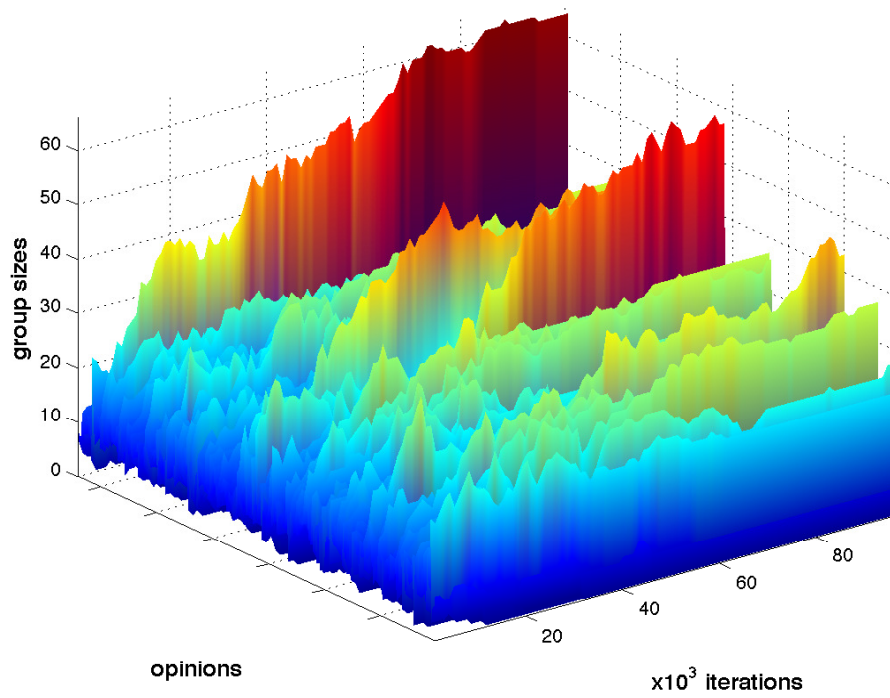


Figure 6: Opinion development over time for $\phi = 0.458$

As could be expected using the data in paper [1], this value of ϕ lets some opinion exchange occur but after some time the network degenerates into several medium-sized, disconnected groups. This is caused by the also equally probable destruction of connections in the network that connect agents of different opinions.

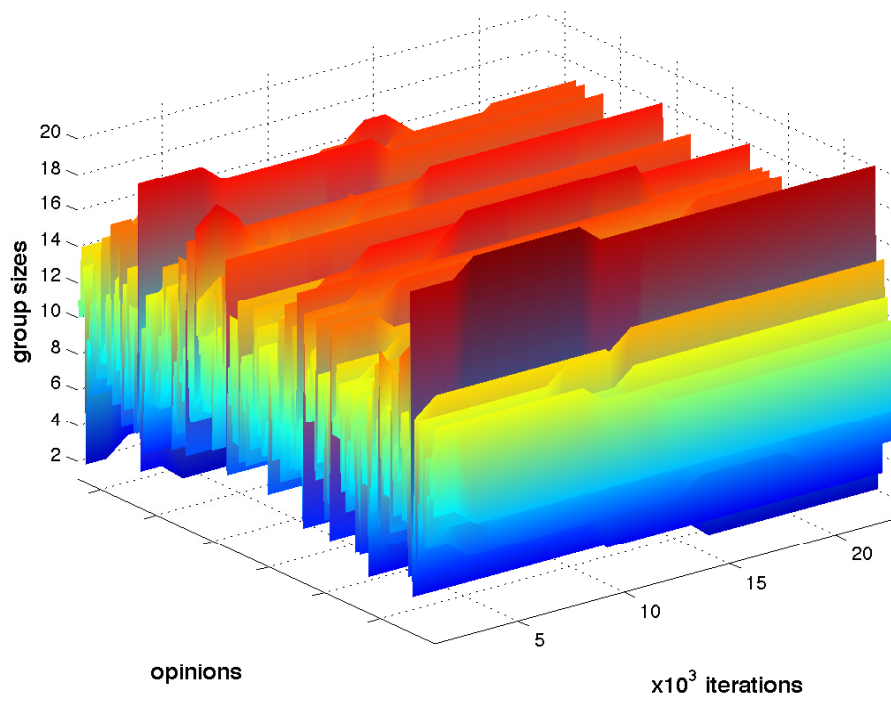


Figure 7: Opinion development over time for $\phi = 0.96$
 Since for such a large ϕ the model mostly connects agents of equal opinions, the actual number of people having a certain opinion stays almost completely unchanged over the course of the simulation.

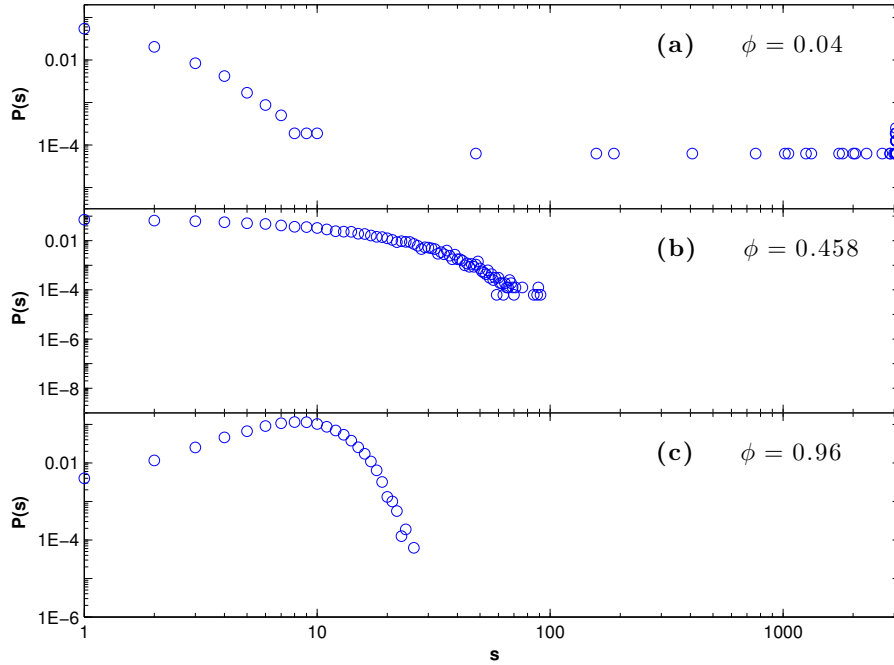


Figure 8: Equivalent to *Figure 2* from paper [1] with our discrete model used for simulation. The figure represents histograms of the community sizes s in the consensus state. The x-axis represents the size of opinion groups, whereas the y-axis specifies proportion of groups of a specific size compared to the number of opinion groups. In principle this is the probability of a certain opinion group size occurring during a simulation. Simulation parameters as in Table 3, except that the data was averaged over 50 simulations. Simulations for $\phi = 0.04$ and $\phi = 0.458$ were run for $20 \cdot 10^6$ iterations, 10^6 iterations were used for $\phi = 0.96$.

Figure 8 shows the results of our simulations. The figure is formatted identically to *Figure 2* from paper [1] to allow straightforward comparisons. Ideally these two figures would be identical but as we have already described, we could not run the simulations with exactly the same parameters. While the figures have a very similar overall structure, there also are some notable differences. We will discuss the results for each panel (a), (b) and (c) separately.

(a) $\phi = \mathbf{0.04}$

In general, our results show the same structure and tendencies as the simulations from paper [1]. But most notably, opinion group size varies far more for small $P(s)$ in our simulations. This is quite probably the effect of our insufficient number of iterations. In paper [1] most groups are of size 10 and smaller while a very small number consists of almost all the people. Since we were unable to run the simulation for a sufficiently long time, many simulations had not yet reached consensus and still had a few large competing opinion groups. These groups would probably coalesce if allowed more time and all reach a similar large size. Also our distribution values $P(s)$ don't get as small as in paper [1]. But since $P(s)$ is essentially the probability of an opinion group to reach a specific size, this isn't surprising. In contrast to paper [1] we only averaged over 50 simulations instead of 10^4 . Obviously only one group per simulation can reach a size approaching N so the probability of such large groups decreases heavily as we average over more simulations.

(b) $\phi = \mathbf{0.458}$

This panel seems to show the largest difference when compared. The distribution of community sizes does approximately match the values from paper [1] for the resulting size and distribution combinations. On the other hand, neither do we get such low probabilities nor the correspondent larger group sizes. Again, the lack of small $P(s)$ can be justified by the smaller number of averaging simulations. But while the lack of large group sizes can be partially explained by their rather low probability, we still expected at least a few outliers. The group sizes seem to hit a ceiling at about size 100. It could be that our model somehow favors the disconnection of opinion groups and therefore the formation of more smaller, disconnected opinion communities. But we were unable to ascertain the exact reason for this unexpected behaviour.

(c) $\phi = \mathbf{0.96}$

Panel (c) seems to be almost identical. Our simulation seems to produce the Poisson distribution with mean λ . This was expected, as we were able to run more than enough iterations to reach consensus state and since for such large ϕ the original random distribution is mostly preserved. Once more the lack of very small values of $P(s)$ can most probably be explained by the smaller number of simulations that we averaged about.

While our recreated model does not exactly match the original networked model from paper [1], our model still produces very similar results and we therefore decided to use it as the basis for our later efforts.

3.1.2. Watts and Strogatz network

In the previous subsection we ran all the simulations with a random graph as the initial social structure. Now we change the underlying network to the Watts and Strogatz model and examine the similarities and differences. To compare the results to the previous ones, we used the same simulation parameters. The simulation is run with $N = 3200$ agents, each one having $k = 4$ neighbours and the probability value ϕ has the values 0.04, 0.458 and 0.96 whereas the opinion ratio γ is 10.

The important parameter for the Watts and Strogatz model is β , which describes whether we have a regular network structure with local clustering $\beta \rightarrow 0.0$ or a random graph $\beta \rightarrow 1.0$. Figure 9 shows the simulation like in Figure 8 but now with the Watts and Strogatz model for $\beta = 0.25$. That each agent has four neighbours $\beta = \frac{1}{4}$ means that on average three of the four neighbours are local ones and one is completely randomly chosen from all agents.

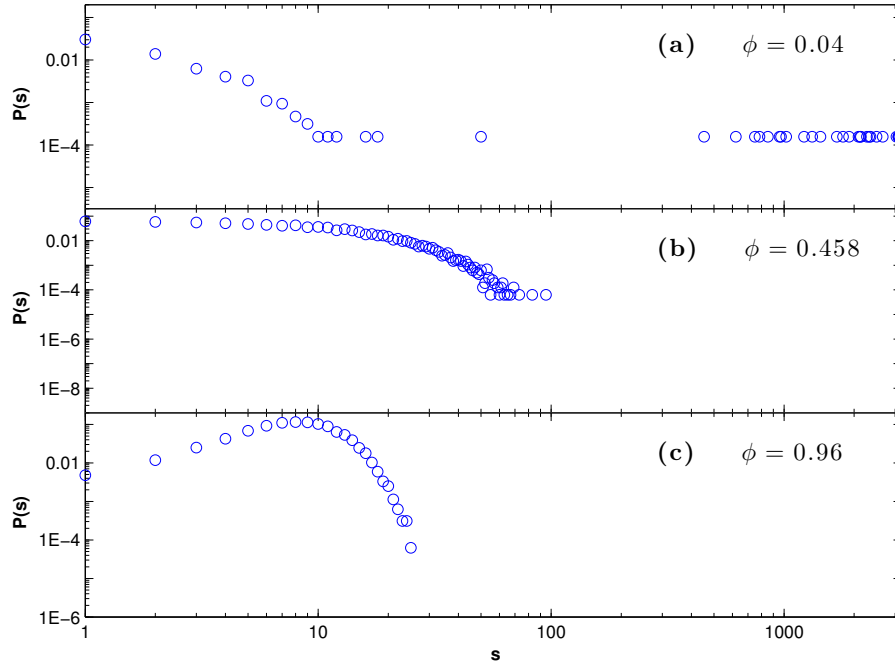


Figure 9: Equivalent to *Figure 2* from paper [1]. Computed using our discrete model with a Watts Strogatz network. See Figure 8 for more information and parameter values. The value of β was set to 0.25.

We observe no remarkable difference between Figure 9 and Figure 8 and therefore studied the impact of different values of β .

To examine the dependency on β we run the simulation for different values of β . Figure 10 shows the simulation for $\beta = 0.0$, Figure 11 for $\beta = 0.5$ and Figure 12 for $\beta = 1.0$. For each Figure the simulation was run 10 times and shows the average over these runs.

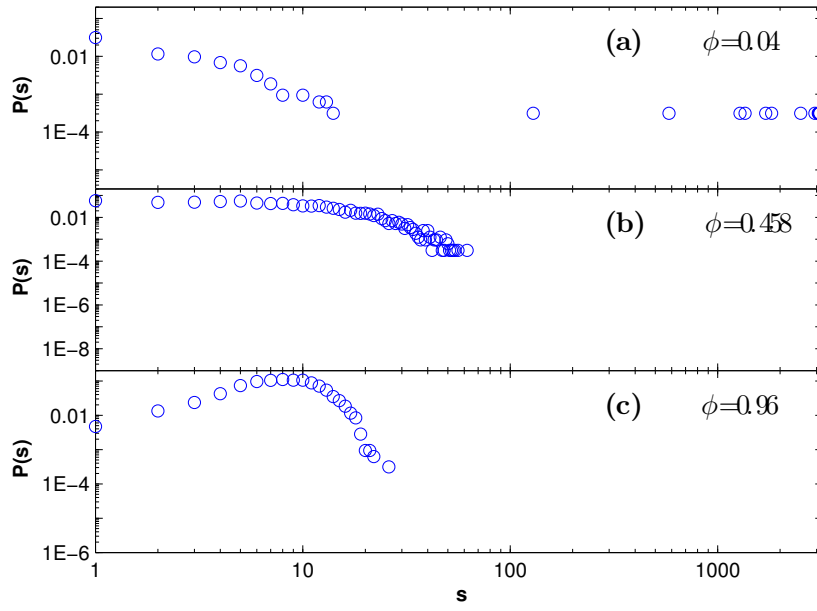


Figure 10: Opinion group size distribution with Watts and Strogatz model and $\beta = 0.00$

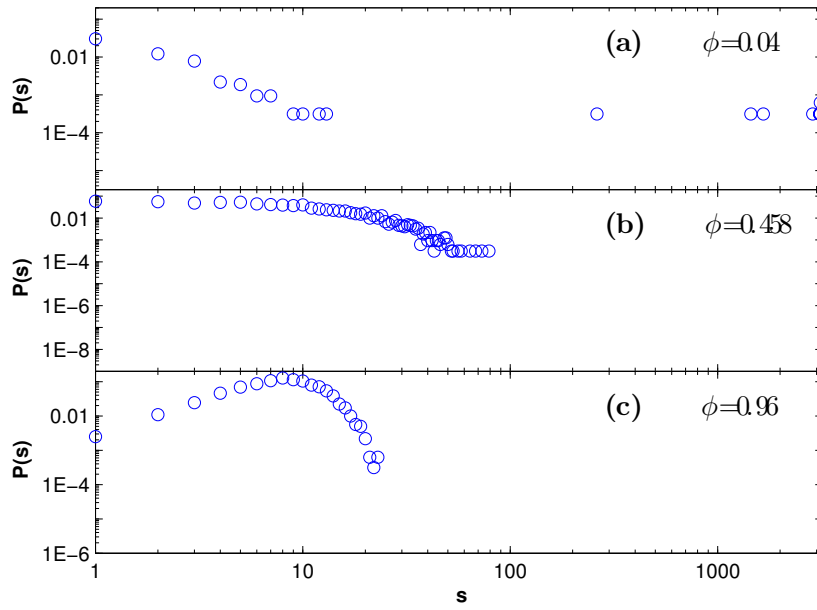


Figure 11: Opinion group size distribution with Watts and Strogatz model and $\beta = 0.50$

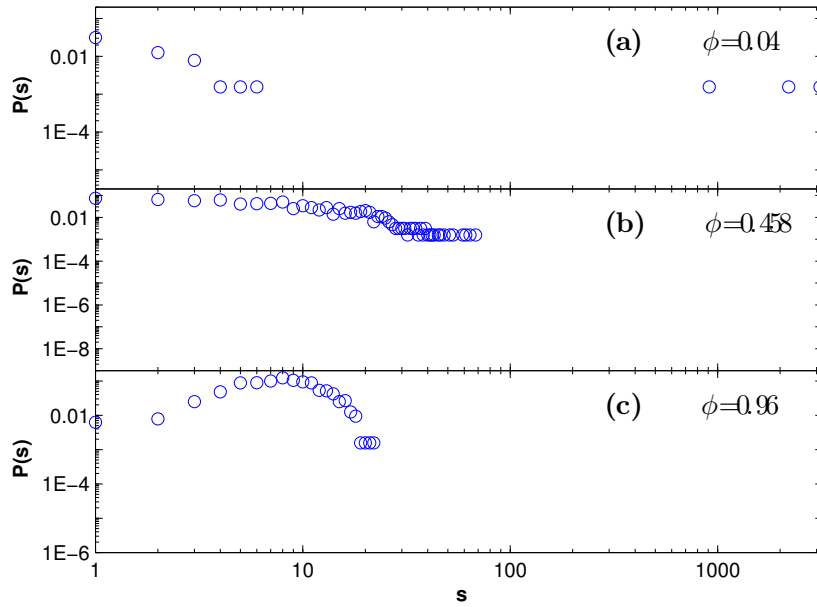


Figure 12: Opinion group size distribution with Watts and Strogatz model and $\beta = 1.00$

As can be observed in Figures 10, 11 and 12, there is no notable difference in the opinion group size distribution for varying values of β . This allows us to answer our first research question. We determined that whether we choose a random graph or the Watts and Strogatz graph as the initial network does not influence the distribution of opinion group sizes in the given model. Therefore a random graph performs well as an approximation of social structures.

We explain that there is no difference whichever network structure is chosen because of the way the discrete model works. In step one of the model (see section 2.1.1) one of the neighbours of the agent is replaced by a random agent holding the same opinion. Because the opinions are initially distributed randomly an initial regular network structure does not influence the final opinion group size distribution. This is because the regular structure is broken up during the iterations of the simulation by the above described step one of the model, which introduces links to random agents. So during the simulation the regular structure changes to a random one before the consensus state is reached.

Although we could not notice an influence of the network structure on the opinion group size distribution we noticed a difference in the number of iterations required to fall below the specified threshold for different values of beta. So we did not run the simulation with a fixed number of iterations but with a threshold of 0.05 (see section 2.3.1). Figure 13 shows the number of iterations for the three values of ϕ . The values were measured for $\beta = 0.0$, $\beta = 0.5$ and $\beta = 1.0$.

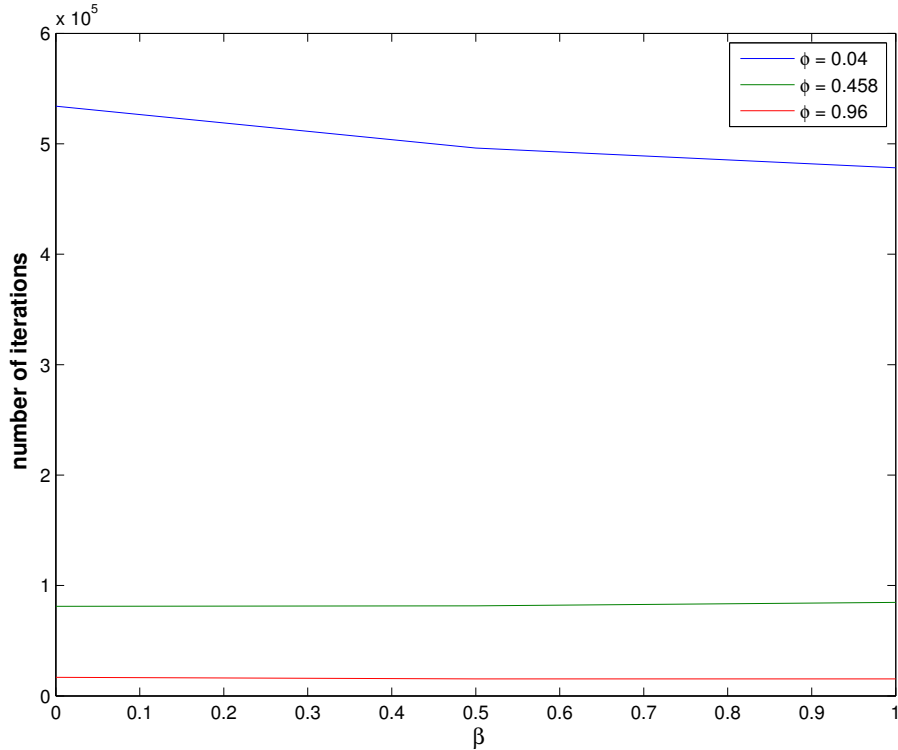


Figure 13: Number of iterations required to fall below threshold 0.05 for $\beta = 0.0$, $\beta = 0.5$ and $\beta = 1.0$

For $\phi = 0.458$ and $\phi = 0.96$ no striking dependency on β can be observed. For $\phi = 0.04$ however the number of iterations goes down for increasing β . This corresponds nicely with the above explanation. For $\phi = 0.04$ *Method 1* of the discrete model is chosen with low probability, therefore it takes longer to break up the regular structure and reach the consensus state.

3.2. Continuous network-based model

3.2.1. Behaviour of the continuous model implementation

The formation of different opinion groups was studied in more detail for the continuous network based model⁶. The continuous, not network-based model presented in [2] was known to have the following behaviour.

1. Consensus can only be achieved for $u \geq 0.3$.
2. The number of different opinion groups is defined by the integer part of the expression $\frac{1}{2u}$.

The parameter u is the opinion threshold of the continuous model presented in section 2.1.2 and is used in the opinion transformation (1). Two agents meeting will only converge in their opinion if it differs by less than u . The continuous model is a special case of the combined model when setting $\phi = 0$ and using a complete graph as the social network structure. We found our implementation to almost match the specification given in the paper, varying in only ± 1 group. The result is shown in table 2. Although some of the simulation parameters were specified in [2] we used different ones as the simulation would have taken too much computational effort.

⁶ called *combined model* from now on to prevent confusion

Threshold u	0.3	0.25	0.1	0.0625	0.0333
Expected Groups	1.67	2	5	8	15
Continuous Model	1	2	4	7	15

Table 2: Number of opinion groups in the continuous model implementation.

3.2.2. Simulation setup for the combined model

To study the convergence of the combined model, most of the parameters were fixed and only few were changed. The setup aimed to be similar to the one in the discrete model. The parameters are shown in Table 3.

Parameter	Value	Description
N	3200	Number of agents in the model
k	4	Number of connections of each agent, i.e. number of other agents know to each one
clusters	320	Number of opinion clusters to extract from the continuous range. This matches the $\gamma = 10$ setting in the discrete model.
std	0.1	Standard distribution of probability to find like-minded people.
μ	0.3	Convergence parameter.
iterations	1000000	Number of iterations to perform.
cskip	100	Number of iterations to summarize in one entry of the histogram.

Table 3: Simulation setup for the combined model.

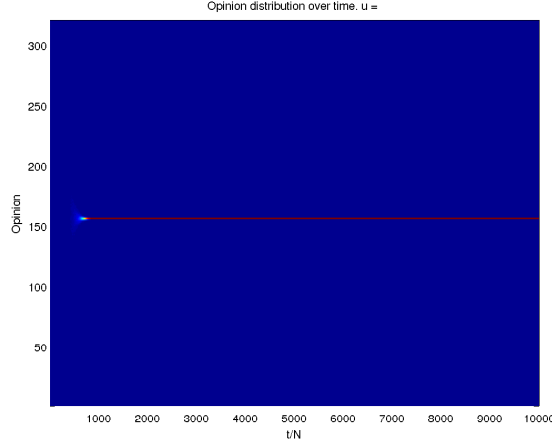
The varying settings are the underlying graph and the parameter ϕ describing the probability of choosing *Method 1 or 2* of the agent interaction in each simulation step as already known from the discrete model. For ϕ we chose the values 0.040, 0.458 and 0.960 as in the discrete model. To model the social network, the random graph, the Watts and Strogatz graph with $\beta = 0.25$ and the Watts and Strogatz graph with $\beta = 0.50$ were chosen.

3.2.3. Group formation behaviour and interpretation of the combined model

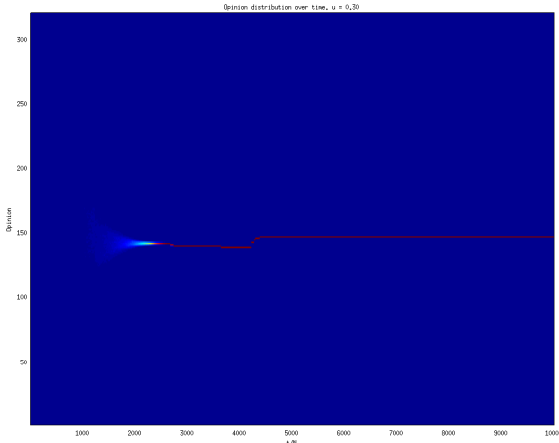
The incorporation of a network structure into the continuous model leading to the combined model had a huge impact on the opinion formation in the system. Research question 2a was therefore revealed to be the most interesting one and is discussed in detail in this section. Research questions 2b and 2c will be discussed when appropriate in the discussion of research question 2a.

In section 3.1.2 was explained that the Watts and Strogatz graph compared to the random graph did not have an notable impact on the result for the discrete model. The same was found for the combined model and the explanation from section 3.1.2 applies also for the combined model. So research question 2b does not reveal any notable result.

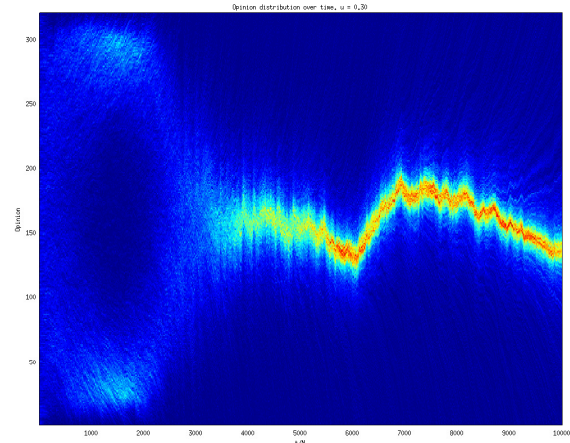
The combined model was found to form disjoint opinion groups with the same pattern $count(u) = \frac{1}{2u}$ as the continuous model. However the convergence differs hugely based on the chosen parameters. Furthermore, even though the opinion groups separate themselves from each other, they tend to not really agree on a specific opinion but accept a much broader range of opinions in one single group.



(a) Continuous Model



(b) Combined Model, $\phi = 0.040$

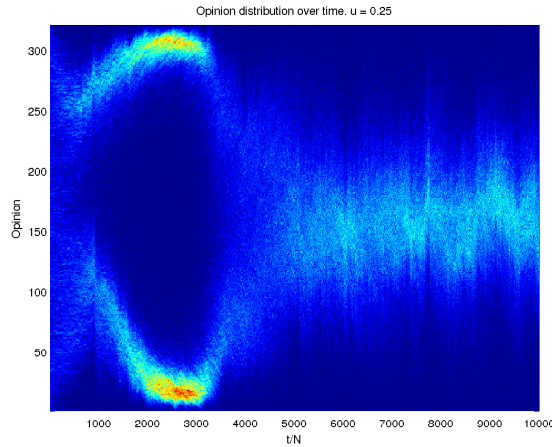


(c) Combined Model, $\phi = 0.458$

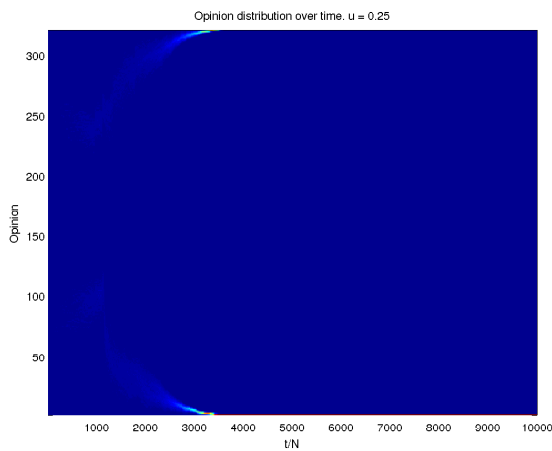
Figure 14: Convergence behaviour from continuous and combined model.

Note that for all graphs the same simulation settings were chosen. The count of iterations on the x-axis has to be multiplied by 100 as `cskip = 100`.

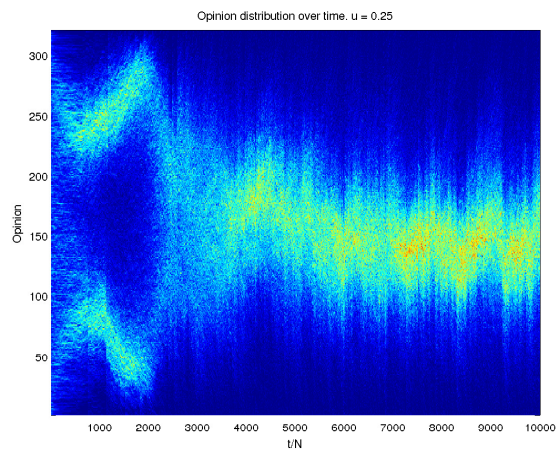
One can see that the continuous model in Figure 14a converges very fast in less than 100 000 iterations. The combined model in Figure 14b also has a very clean look, but it takes 250 000 iterations to converge. This can easily be understood as the low value of $\phi = 0.040$ makes the model behave similar to the continuous model. It should be mentioned though, that while the continuous model considers interactions of any two agents in the system, in the combined model only adjacent agents will meet in *Method 2* of the model and converge in their opinion. This might be an explanation for the glitch in Figure 14b around the 450 000th iteration. The effect becomes even stronger when giving the network structure more influence by setting $\phi = 0.458$. See Figure 14c. The opinion range in the single group is much broader as well. The impact of the network structure will be examined further now. Let us consider the graphs in Figure 15.



(a) Example of a very unstable decision



(b) Example of a clear decision



(c) Opinion groups cannot separate

Figure 15: Stability of the decision in the combined model for $\phi = 0.040$ and $\frac{1}{2u} \approx 2$

Given a small value of ϕ the behaviour on higher u , so that only one or two opinion groups are expected, is very unstable. We assume this is due to the very few changes in the network structure. With a small ϕ the second method is almost always chosen. This method only lets agents change their opinion in their given network which itself remains almost static. Only the first method will change the network structure. If there are two almost disjoint components in the social network, they can easily evolve independent opinions and separate themselves from each other. This can be seen in Figure 15b. Exactly the opposite is the case if the network is heavily connected. In this case no opinion can separate itself from the global network. As every agent is only connected a small number of other agents, its opinion cannot change very strongly. On the other hand, each of its neighbours is also connected to other neighbours that can hardly change their opinion as well. Thus the opinion range in this giant community doesn't converge into a small domain as Figure 15c demonstrates. The combination of these two behaviours can lead to a very unstable system as one can see in Figure 15a.

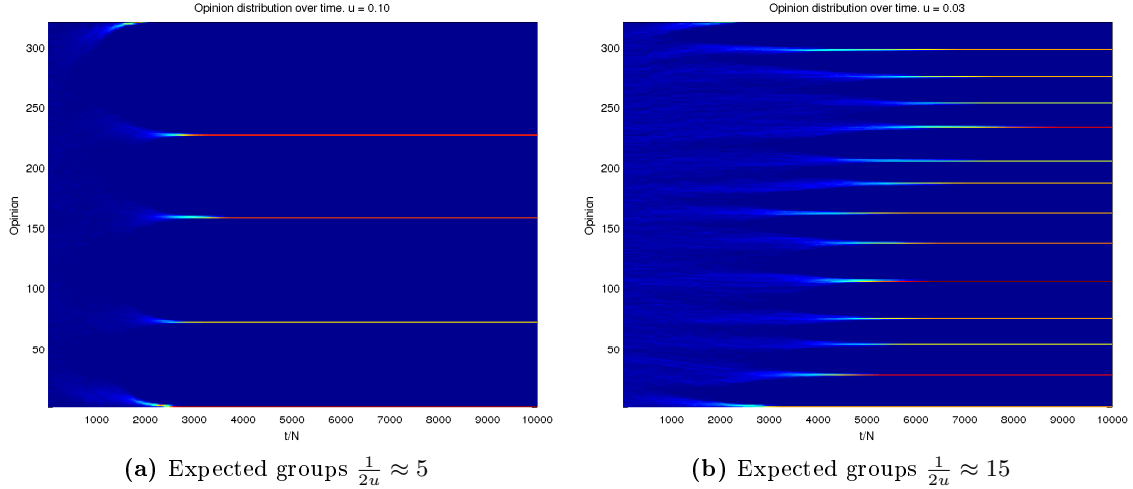


Figure 16: Influence of $\phi = 0.458$ in the combined model

For $\phi = 0.458$ the results look well-behaved as pictured in Figure 16. Disjoint groups persist once formed. Figure 16 also shows that the rule for the number of disjoint groups $count(u) = \frac{1}{2u}$ applies to the combined model, which answers research question 2c. In Figure 16a the expected number of groups is 5 which is also the value we observe. In Figure 16b we expect 15 groups and count 14, which is off by one. Generally, the lower u is chosen, the more opinion groups will be formed and the convergence is much more stable as each little group may persist. It is only for higher values of u that the behaviour is more dynamic. As well, with $\phi = 0.458$ both *Method 1* and *2* are selected with almost equal probability which is a good mixture of network and opinion change. With a higher value for ϕ the model chooses *Method 1* more often which leads to a very chaotic behaviour examined in the next paragraph.

A mid-range ϕ provided nice results regarding convergence. A high value such as $\phi = 0.960$ debases the convergence quality. Even compared to a low-range ϕ , choosing ϕ to be close to one leads to a very bad convergence, as Figure 17 shows. Still the $count(u) = \frac{1}{2u}$ seems to hold. One can guess that Figure 17a will lead to two or three and Figure 17b will lead to six opinion groups. The reason for this can be found in the rare change of opinion in the whole system. In opposition to a system with a low-range ϕ explained before, for a high-range ϕ *Method 1* is chosen very often. This leads to a highly dynamic network. But the agents in the system do not change their opinion often as the probability for the second method is low. The system consists of agents constantly changing their network but not interacting with each other.

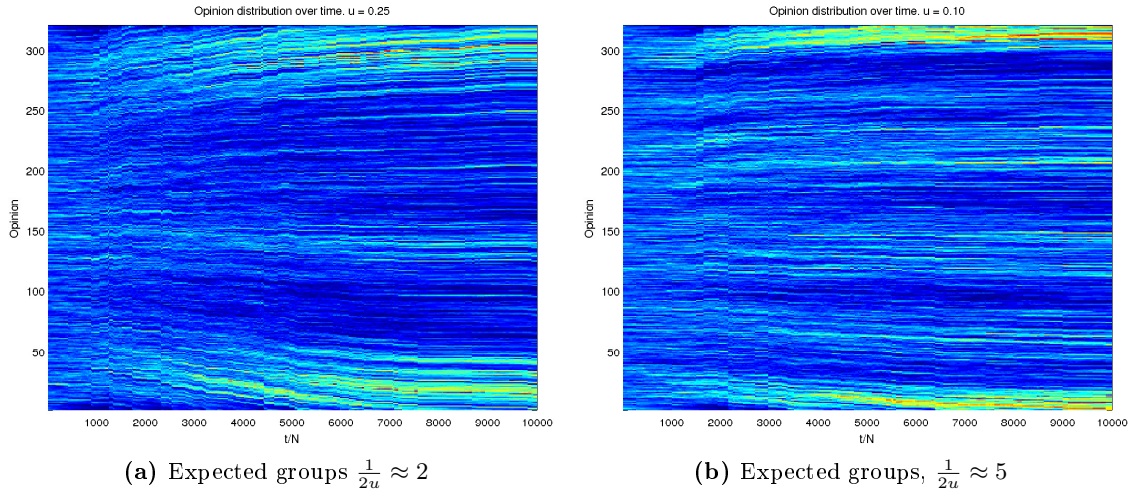


Figure 17: Influence of a high ϕ in the combined model for $\phi = 0.960$

4. Summary and Outlook

To summarize, we have reproduced the discrete network-based model presented in paper [1] and extended it with the ideas from the continuous model from paper [2]. While we were able to replace the initial network structure with a small world structure and enhance the network-based model with continuous values, further research into those areas is necessary to fully understand their impacts.

Apart from a few smaller issues we were mostly able to recreate the discrete network-based model proposed in paper [1]. Our implementation produces simulation data that shows the same patterns as the model from paper [1]. This allowed us to investigate the model further using several modifications. We exchanged the random initial network with a more sophisticated small world structure and combined the network approach with the continuous model proposed in paper [2]. While our discrete model seems to be working mostly correct a few uncertainties remain. Especially the lack of larger group sizes for $\phi = 0.458$. With more time and computation power, we could examine our discrete model more closely, primarily running the simulation for the proposed 10000 times. In this way we could further assert the correctness of our implementation. Furthermore, the behaviour for $\phi = 0.458$ could be analysed more closely to determine the cause of the group size limit we observed.

We did not find an influence when using a Watts and Strogatz graph instead of a random one. The distribution of the opinion group sizes in the discrete network-based model showed no notable difference. Likely this is partly due to the fact that the initial opinions are distributed randomly and do not depend on the local structure. The opinion development with the Watts and Strogatz model could be analysed further especially at the beginning of the simulation. Moreover it would be reasonable not to distribute the initial opinions randomly but take the underlying network into account and form local clusters.

Another interesting parameter worth mentioning is k , the number of neighbours of an agent. In our simulations we fixed the value at 4 as proposed in paper [1]. It is left to further examination whether or not the network has an influence on achieving consensus for greater values of k .

The combined model was found to behave pretty much the same as the discrete and the contin-

uous models in the studied characteristics regarding the final state of consensus. Different initial network structures did not change the overall behaviour of the combined model. The opinion groups matched the already known formula $count(u) = \frac{1}{2u}$ from the continuous model. Nevertheless, achieving the consensus state is heavily dependent on the parameter ϕ . It greatly impacts the influence of the network and limits agents to only converge towards connected agents instead of the whole network as in the continuous model. Especially for a high opinion threshold u , for which we expected a small amount of opinion groups, the system showed to be very unstable. The social network of each agent seems to unexpectedly change the single opinion towards which the whole system is converging (see Figure 14c). On the other hand the connected agents could not separate into different opinion groups as seen in Figure 15c.

Combining the network structure with the continuous model revealed the unexpected effect of oscillating opinions. Even though the system converged to a single opinion group, that very opinion moved up and down over the whole opinion domain. We could only guess that this is due the network structure as we could not observe this behaviour in any of our simulations of the continuous model. Taking this into account, studying the instability of the system even further would be a very interesting project. The combined model offers even more aspects to investigate further. *Method 1* chooses a new neighbour with a normal distributed probability. During our research, the standard derivation `std` of that probability was fixed to 0.1. The influence of a different standard derivation on achieving consensus has not been studied. It would also be possible to compare the discrete and the combined model in more detail. The code basis for this has been provided in this project. `OUTPUT 1-3` in the section of the combined model in the file `generateGraphs.m` (Listing 12 (Lines 891-1157)) generates graphs similar to the ones of the discrete model and can be used as a starting point for further investigations.

Although a lot a reseach was done on opinion formation is seems that there are still a lot of aspects that remain insufficiently examined. To understand the complex phenomena of opinion formation these will have to be investigated further.

A. References

- [1] Petter Holme and M. E. J. Newman. Nonequilibrium phase transition in the coevolution of networks and opinions. *Phys. Rev. E*, 74(5), Nov 2006.
- [2] M. F. Laguna, Guillermo Abramson, and Damián H. Zanette. Minorities in a model for opinion formation: Research articles. *Complex.*, 9(4):31–36, 2004.
- [3] WIKIPEDIA. Random graph. http://en.wikipedia.org/wiki/Random_graph, Jul 2010.
- [4] WIKIPEDIA. Watts and Strogatz model. http://en.wikipedia.org/wiki/Watts_and_Strogatz_model, Okt 2010.

B. List of Figures

1.	Different network structures for $n = 20$ and $k = 4$. Source: http://www.cmt.phys.kyushu-u.ac.jp/kenkyu_syokai_en/NeuralNetworks/images/wsnetwork.png	6
2.	Convergence of model $\phi = 0.04$	14
3.	Convergence of model $\phi = 0.458$	15
4.	Convergence of model $\phi = 0.96$	15
5.	Opinion development over time for $\phi = 0.04$ We can see that as expected the opinions change intensively. At the beginning most opinion groups have sizes between 0 and approximately 100 with a mean of $\gamma = 10$. Very quickly several larger groups emerge. When the simulation is run long enough, eventually the largest opinion group dominates and absorbs all agents in the system it can connect to. When consensus is reached only one giant community is left.	16
6.	Opinion development over time for $\phi = 0.458$ As could be expected using the data in paper [1], this value of ϕ lets some opinion exchange occur but after some time the network degenerates into several medium-sized, disconnected groups. This is caused by the also equally probable destruction of connections in the network that connect agents of different opinions.	17
7.	Opinion development over time for $\phi = 0.96$ Since for such a large ϕ the model mostly connects agents of equal opinions, the actual number of people having a certain opinion stays almost completely unchanged over the course of the simulation.	18
8.	Equivalent to <i>Figure 2</i> from paper [1] with our discrete model used for simulation. The figure represents histograms of the community sizes s in the consensus state. The x-axis represents the size of opinion groups, whereas the y-axis specifies proportion of groups of a specific size compared to the number of opinion groups. In principle this is the probability of a certain opinion group size occurring during a simulation. Simulation parameters as in Table 3, except that the data was averaged over 50 simulations. Simulations for $\phi = 0.04$ and $\phi = 0.458$ were run for $20 \cdot 10^6$ iterations, 10^6 iterations were used for $\phi = 0.96$	19
9.	Equivalent to <i>Figure 2</i> from paper [1]. Computed using our discrete model with a Watts Strogatz network. See Figure 8 for more information and parameter values. The value of β was set to 0.25.	21
10.	Opinion group size distribution with Watts and Strogatz model and $\beta = 0.00$	22
11.	Opinion group size distribution with Watts and Strogatz model and $\beta = 0.50$	22
12.	Opinion group size distribution with Watts and Strogatz model and $\beta = 1.00$	23
13.	Number of iterations required to fall below threshold 0.05 for $\beta = 0.0$, $\beta = 0.5$ and $\beta = 1.0$	24
14.	Convergence behaviour from continuous and combined model.	26
15.	Stability of the decision in the combined model for $\phi = 0.040$ and $\frac{1}{2u} \approx 2$	27
16.	Influence of $\phi = 0.458$ in the combined model	28
17.	Influence of a high ϕ in the combined model for $\phi = 0.960$	29

C. List of Tables

- 1. Parameters used in paper [1] to create *Figure 2*. 13
- 2. Number of opinion groups in the continuous model implementation. 25
- 3. Simulation setup for the combined model. 25

D. MATLAB[®]code

D.1. continuousModel.m

Listing 8: continuousModel.m

```
1 function [people, opinions, histogram] = ...
2     continuousModel(people,opinions,phi,u,mu,std,iter,clusters,cskip)
3     % The combined Model.
4 % INPUT
5 % people    adjacency matrix representing the system
6 % opinions  vector of the opinions of the agents
7 % phi       probability factor for model
8 % u         opinion threshold
9 % mu        convergence parameter
10 % std       standard derivation used in probability to find
11 %           like-minded people
12 % iter      number of iterations to perform
13 % clusters  how many opinion clusters the histogram should contain (u0)
14 % cskip    skip factor for the returned histogram with respect to
15 %           the number of iterations
16 % OUTPUT
17 % people    adjacency matrix after the simulation
18 % opinions  opinion vector after the simulation
19 % histogram this is a ( clusters+1 x floor(iter/skip) ) matrix.
20 %           the first #clusters rows contain the count of people
21 %           holding an opinion in that cluster. the j-th column
22 %           represents the state after (j-1)*cskip iterations.
23 %           the last row contains the norm of the change in the
24 %           clusterization process.
25
26 % Number of people in the system
27 n = length(opinions);
28
29 % histogram stores the count of poeple holding the opinion in each cluster
30 % and the norm in clusterization change.
31 histogram = zeros(clusters + 1,floor(iter/cskip));
32
33 % in order to find people with a similiar opinion, a sorted copy
34 % of the opinions is created with a link to the original person
35 % index: sorted_opinions = [ opinion , index ]
36 sorted_opinions = [opinions , (1:n)'];
37 sorted_opinions = sortrows(sorted_opinions);
38
39 old_hist = zeros(clusters,1);
40 % Main loop, iterating over t
41 for t=1:iter
42
43     % print progress
44     if exist('DEBUG')
45         if mod(t,50000) == 0
46             fprintf('%i/%i\n',t,iter);
47         end
48     end
49
50     % update histogram
51     if mod(t,cskip) == 0
52         h = hist(opinions,clusters)';
53         histogram(:,t/cskip) = [ h; norm(old_hist - h) ];
54         old_hist = h;
```

```

55     end
56
57     % Select a person randomly
58     person = randi(n);
59
60     % Check if the person is still connected to someone
61     % Do nothing otherwise
62     deg = sum(people(person,:));
63     if deg == 0, continue; end
64
65     % Find all neighbours connected to that person
66     neighbours = find(people(person,:));
67
68     % Select a neighbour at random
69     j = randi(length(neighbours));
70     neighbour = neighbours(j);
71
72     % Chose method 1 or 2 with probability phi
73     if rand() <= phi
74         % METHOD 1
75         % Select at random one of the edges attached to person
76         % and move the other end of that edge to a another person.
77         % The new neighbour is choosen by a probability that is
78         % normally distributed according to the difference in
79         % opinion of the possibly new neighbour.
80
81         % figure out what opinion the new neighbour should have
82         neighbours_opinion = opinions(person) + std*randn();
83         % do not accept opinions out of range [0,1]
84         % try to find another person
85         acceptable = 0;
86         for i=1:100
87             if neighbours_opinion < 0 || neighbours_opinion > 1
88                 neighbours_opinion = opinions(person) + std*randn();
89             else
90                 acceptable = 1;
91                 break;
92             end
93         end
94         % if that person has such an extreme opinion that it can't find
95         % anyone within range [0,1], the extrema is taken
96         if acceptable == 0
97             if neighbours_opinion < 0
98                 neighbours_opinion = 0;
99             elseif neighbours_opinion > 1
100                 neighbours_opinion = 1;
101             end
102         end;
103
104         % find the person with the closest opinion to
105         % neighbours_opinion
106         i = find(sorted_opinions(:,1) > neighbours_opinion, 1 , 'first');
107         if size(i,1) == 0
108             % no one has such a strong (-> 1) opinion.
109             % Take the person with the strongest opinion
110             new_neighbour = sorted_opinions(end,2);
111         elseif i == 1
112             % there is no person with a lower opinion, so the person
113             % at index i is the new neighbour
114             new_neighbour = sorted_opinions(i,2);
115         else

```

```

116         % the person at index i-1 might be closer to the opinion
117         diff_n1 = neighbours_opinion - sorted_opinions(i,1);
118         diff_n2 = neighbours_opinion - sorted_opinions(i-1,1);
119         if abs(diff_n1) < abs(diff_n2)
120             new_neighbour = sorted_opinions(i,2);
121         else
122             new_neighbour = sorted_opinions(i-1,2);
123         end
124     end
125
126     % Connect person and new_neighbour
127     % Disconnect person and old_neighbour
128     if person ~= new_neighbour
129         people(person,new_neighbour) = 1;
130         people(new_neighbour,person) = 1;
131         people(person,neighbour) = 0;
132         people(neighbour,person) = 0;
133     end
134 else
135     % METHOD 2
136     % Pick a random neighbour of person. If their opinion
137     % differ by more than the opinion threshold u, do nothing.
138     % Else, converge the two opinions according to the
139     % convergence parameter mu.
140
141     diff = opinions(neighbour) - opinions(person);
142     if( abs( diff ) <= u )
143         % update opinion vector
144         opinions(person) = opinions(person) + mu*diff;
145         opinions(neighbour) = opinions(neighbour) + mu*(-diff);
146         % pass update to sorted vector
147         altPerson = sorted_opinions(:,2) == person;
148         altNeighbour = sorted_opinions(:,2) == neighbour;
149         sorted_opinions(altPerson,1) = opinions(person);
150         sorted_opinions(altNeighbour,1) = opinions(neighbour);
151         sorted_opinions = sortrows( sorted_opinions );
152     end
153 end
154
155 end
156
157 end

```

D.2. createRandomSocialGraph.m

Listing 9: createRandomSocialGraph.m

```

1 function [ M ] = createRandomSocialGraph( n, k )
2 %createRandomSocialGraphs
3 % Create a random sparse adjacency matrix of size (n,n).
4 % k specifies the average degree of each node.
5 % n : Number of people
6 % k : Number of people known to each person
7
8 if nargin < 2, error('Insufficient input arguments.');
```

```

13
14 % Generate a random symmetric matrix with given density
15 M = sprandsym(n, density);
16
17 % Set all nonzero entries to 1
18 nz = find(M);
19 M(nz) = 1;
20
21 end

```

D.3. createWattsAndStrogatzModel.m

Listing 10: createWattsAndStrogatzModel.m

```

1 function [ A ] = createWattsAndStrogatzModel( n, k, beta )
2 %           watts and strogatz model
3 % Create a sparse matrix A representing a undirected graph in the Watts
4 % and Strogatz model
5 % see: http://en.wikipedia.org/wiki/Watts\_and\_Strogatz\_model
6 %
7 % Input:  n      number of nodes
8 %         k      mean degree k
9 %         beta   interpolation parameter between ER graph and regular
10 %              ring lattice (0 <= beta <= 1)
11 %         beta = 0 : regular ring lattice
12 %         beta = 1 : ER graph
13 % Output: A      sparse matrix
14 %
15 % the parameters must satisfy n >> k >> ln(n) >> 1
16
17 % Use default values if no input is given
18 if nargin < 2
19     k = 20;
20     beta = 0.5;
21 end
22
23 % error handling
24 if (mod(k, 2) == 1)
25     error('The mean degree k must be divisable by 2.');
```

```

46 % For every node (row in the matrix) take every edge (n_i, n_j) with i < j
47 % and rewire it with probability beta. I.e. we replace (n_i, n_j) with
48 % (n_i, n_r). (step 2 wikipedia)
49 for i=1:n
50     neighbours = find(A(i,:));
51     for j=neighbours
52         if (i < j) && (rand < beta)
53             r = i;
54             % choose until candidate found that avoid loops and link duplication
55             while ( (r == i) || (full(A(i,r)) == 1) )
56                 r = randi(n); % choose random integer number
57             end
58             A(i,j) = 0; A(j,i) = 0; % reset edge
59             A(i,r) = 1; A(r,i) = 1; % new edge
60         end
61     end
62 end
63
64 % consistency check
65 if (nnz(A) ~= k*n)
66     error(['The number of edges must be k*n/2 and therefore the number'...
67         'of nonzero elements n*k']);
68 end
69
70 end

```

D.4. generateContinuousOpinions.m

Listing 11: generateContinuousOpinions.m

```

1 function opinions = generateContinuousOpinions( n )
2 %generateContinuousOpinions
3 % Generate a random opinion vector of length n with continuous
4 % opinions in the range [0,1].
5 opinions = rand(n,1);
6 end

```

D.5. generateGraphs.m

Listing 12: generateGraphs.m

```

1 % Generate graphs for project report
2
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 % GRAPHS FOR THE NETWORK-BASED MODEL
5
6 %% OUTPUT 1a - Model correctness - Random
7 % Equivalent figure to "Figure 2" in paper 1
8 % Used to compare discrete model against model from paper 1
9
10 clear all;
11 close all;
12 clc;
13
14 % Set parameters
15 n = 3200;
16 k = 4;

```

```

17 gamma = 10;
18 average_iterations = 50;
19 iter = 20000000;
20 figureFolder = '../documentation/figures/';
21
22
23 fprintf('Random\n');
24
25 % Generate values
26 phi = 0.04;
27 fprintf('Part 1: phi = %1.3f\n', phi);
28 dist1 = runModel(n,'random',k,gamma,phi,iter,average_iterations);
29
30 phi = 0.458;
31 fprintf('Part 2: phi = %1.3f\n', phi);
32 dist2 = runModel(n,'random',k,gamma,phi,iter,average_iterations);
33
34 phi = 0.96;
35 fprintf('Part 3: phi = %1.3f\n', phi);
36 dist3 = runModel(n,'random',k,gamma,phi,iter,average_iterations);
37
38 % Plot graphic
39 subplot('Position',[.1 .1+1.6/3 .8 .8/3])
40 loglog(0:n-1,dist1./(n/gamma),'o')
41 xlim([0 3200])
42 ylim([10^(-5.5) 0.2])
43 set(gca,'XTickLabel',{})
44 set(gca,'YTick',[0.00001 0.0001 0.001 0.01 0.1])
45 set(gca,'YTickLabel','|1E-4| |0.01| ')
46 ylabel('\bf P(s)')
47 string = '$\textbf{(a)} \quad \phi = 0.04$';
48 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
49     'fontsize',14,'units','norm')
50
51 subplot('Position',[.1 .1+.8/3 .8 .8/3])
52 loglog(0:n-1,dist2./(n/gamma),'o')
53 xlim([0 3200])
54 ylim([10^(-9) 0.2])
55 set(gca,'XTickLabel',{})
56 set(gca,'YTick',[0.0000001 0.000001 0.00001 0.0001 0.001 0.01 0.1])
57 set(gca,'YTickLabel','|1E-8| |1E-6| |1E-4| |0.01| ')
58 ylabel('\bf P(s)')
59 string = '$\textbf{(b)} \quad \phi = 0.458$';
60 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
61     'fontsize',14,'units','norm')
62
63 subplot('Position',[.1 .1 .8 .8/3])
64 loglog(0:n-1,dist3./(n/gamma),'o')
65 xlim([0 3200])
66 ylim([10^(-6) 0.2])
67 set(gca,'YTick',[0.000001 0.00001 0.0001 0.001 0.01 0.1])
68 set(gca,'YTickLabel','|1E-6| |1E-4| |0.01| ')
69 set(gca,'XTickLabel',{'1','10','100','1000'})
70 xlabel('\bf s')
71 ylabel('\bf P(s)')
72 string = '$\textbf{(c)} \quad \phi = 0.96$';
73 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
74     'fontsize',14,'units','norm')
75
76 % Save files
77 filename = [figureFolder,'discrete_model_random'];

```

```

78 print('-dpng', [filename, '.png']);
79 print('-depsc2', [filename, '.eps']);
80
81
82 %% OUTPUT 1b - Small world
83 % Equivalent to OUTPUT 1a, except that
84 % we use the small world system as the initial graph structure
85
86 clear all;
87 close all;
88 clc;
89
90 % Set parameters
91 n = 3200;
92 k = 4;
93 gamma = 10;
94 average_iterations = 20;
95 iter = 10000000;
96 figureFolder = '../documentation/figures/';
97
98 fprintf('Small world\n');
99
100 % Generate values
101 phi = 0.04;
102 fprintf('Part 1: phi = %1.3f\n', phi);
103 dist1 = runModel(n, 'watts_strogatz', k, gamma, phi, iter, average_iterations);
104
105 phi = 0.458;
106 fprintf('Part 2: phi = %1.3f\n', phi);
107 dist2 = runModel(n, 'watts_strogatz', k, gamma, phi, iter, average_iterations);
108
109 phi = 0.96;
110 fprintf('Part 3: phi = %1.3f\n', phi);
111 dist3 = runModel(n, 'watts_strogatz', k, gamma, phi, iter, average_iterations);
112
113 % Plot graphic
114 subplot('Position', [1 1+1.6/3 .8 .8/3])
115 loglog(0:n-1, dist1./(n/gamma), 'o')
116 xlim([0 3200])
117 ylim([10^(-5.5) 0.2])
118 set(gca, 'YTick', [0.00001 0.0001 0.001 0.01 0.1])
119 set(gca, 'YTickLabel', ' |1E-4| |0.01| ')
120 set(gca, 'XTickLabel', {})
121 ylabel('\bf P(s)')
122 string = '$\textbf{(a)} \quad \phi = 0.04$';
123 text('string', string, 'position', [0.7 0.8], 'interpreter', 'latex', ...
124     'fontsize', 14, 'units', 'norm')
125
126 subplot('Position', [1 1+.8/3 .8 .8/3])
127 loglog(0:n-1, dist2./(n/gamma), 'o')
128 xlim([0 3200])
129 ylim([10^(-9) 0.2])
130 set(gca, 'XTickLabel', {})
131 set(gca, 'YTick', [0.00000001 0.0000001 0.000001 0.00001 0.001 0.01 0.1])
132 set(gca, 'YTickLabel', '1E-8| 1E-6| 1E-4| 0.01|')
133 ylabel('\bf P(s)')
134 string = '$\textbf{(b)} \quad \phi = 0.458$';
135 text('string', string, 'position', [0.7 0.8], 'interpreter', 'latex', ...
136     'fontsize', 14, 'units', 'norm')
137
138 subplot('Position', [1 1 .8 .8/3])

```



```

139 loglog(0:n-1,dist3./(n/gamma),'o')
140 xlim([0 3200])
141 ylim([10^(-6) 0.2])
142 set(gca,'YTick',[0.000001 0.00001 0.0001 0.001 0.01 0.1])
143 set(gca,'YTickLabel',{'1E-6| 1E-4| 0.01| '})
144 set(gca,'XTickLabel',{'1','10','100','1000'})
145 xlabel('\bf s')
146 ylabel('\bf P(s)')
147 string = '$\textbf{(c)} \quad \phi = 0.96$';
148 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
149     'fontsize',14,'units','norm')
150
151 % Save files
152 filename = [figureFolder,'discrete_model_ws'];
153 print('-dpng', [filename,'.png']);
154 print('-depsc2', [filename,'.eps']);
155 save([figureFolder,'discrete_model_ws_phi_004.mat']);
156
157 %% OUTPUT 2a - Convergence / Neccessary iterations for consensus - Random
158
159 clear all;
160 close all;
161 clc;
162
163 % Set parameters
164 n = 3200;
165 k = 4;
166 gamma = 10;
167 average_iterations = 1;
168 iter = 0;
169 max_iterations = 100*10^6;
170 threshold = 0;
171 figureFolder = '../documentation/figures/';
172
173
174 % Generate values
175 phi = 0.04;
176 d_it = 10^6;
177 fprintf('Part 1: phi = %1.3f\n', phi);
178 [~,~,~,diffs1] = runModel(n,'random',k,gamma,phi,iter,average_iterations,...
179     max_iterations,threshold,d_it);
180 phi = 0.458;
181 d_it = 10000;
182 fprintf('Part 2: phi = %1.3f\n', phi);
183 [~,~,~,diffs2] = runModel(n,'random',k,gamma,phi,iter,average_iterations,...
184     max_iterations,threshold,d_it);
185 phi = 0.96;
186 d_it = 10000;
187 fprintf('Part 3: phi = %1.3f\n', phi);
188 [~,~,~,diffs3] = runModel(n,'random',k,gamma,phi,iter,average_iterations,...
189     max_iterations,threshold,d_it);
190
191 % Plot graphic & save files
192 plot(diffs1(1,2:end),diffs1(2,2:end),'-')
193 xlabel('\bf iterations','fontsize',12);
194 ylabel('\bf normed difference to last step','fontsize',12);
195
196 filename = [figureFolder,'discrete_model_diff_random_004'];
197 print('-dpng', [filename,'.png']);
198 print('-depsc2', [filename,'.eps']);
199

```

```

200 plot(diffs2(1,2:end),diffs2(2,2:end),'-')
201 xlabel('\bf iterations','fontsize',12);
202 ylabel('\bf normed difference to last step','fontsize',12);
203
204 filename = [figureFolder,'discrete_model_diff_random_0458'];
205 print('-dpng', [filename,'.png']);
206 print('-depsc2', [filename,'.eps']);
207
208 plot(diffs3(1,2:end),diffs3(2,2:end),'-')
209 xlabel('\bf iterations','fontsize',12);
210 ylabel('\bf normed difference to last step','fontsize',12);
211
212 filename = [figureFolder,'discrete_model_diff_random_096'];
213 print('-dpng', [filename,'.png']);
214 print('-depsc2', [filename,'.eps']);
215
216
217 %% OUTPUT 2b - Convergence / Neccessary iterations for consensus - Random
218
219 clear all;
220 close all;
221 clc;
222
223 % Set parameters
224 n = 3200;
225 k = 4;
226 gamma = 10;
227 average_iterations = 1;
228 iter = 0;
229 max_iterations = 100*10^6;
230 threshold = 0;
231 figureFolder = '../documentation/figures/';
232
233
234 % Generate values
235 phi = 0.04;
236 d_it = 10^6;
237 fprintf('Part 1: phi = %1.3f\n', phi);
238 [~,~,~,diffs1] = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations,...
239     max_iterations,threshold,d_it);
240 phi = 0.458;
241 d_it = 10000;
242 fprintf('Part 2: phi = %1.3f\n', phi);
243 [~,~,~,diffs2] = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations,...
244     max_iterations,threshold,d_it);
245 phi = 0.96;
246 d_it = 10000;
247 fprintf('Part 3: phi = %1.3f\n', phi);
248 [~,~,~,diffs3] = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations,...
249     max_iterations,threshold,d_it);
250
251 % Plot graphic & save files
252 plot(diffs1(1,2:end),diffs1(2,2:end),'-')
253 xlabel('\bf iterations','fontsize',12);
254 ylabel('\bf normed difference to last step','fontsize',12);
255
256 filename = [figureFolder,'discrete_model_diff_ws_004'];
257 print('-dpng', [filename,'.png']);
258 print('-depsc2', [filename,'.eps']);
259
260 plot(diffs2(1,2:end),diffs2(2,2:end),'-')

```

```

261 xlabel('\bf iterations','fontsize',12);
262 ylabel('\bf normed difference to last step','fontsize',12);
263
264 filename = [figureFolder,'discrete_model_diff_ws_0458'];
265 print('-dpng', [filename,'.png']);
266 print('-depsc2', [filename,'.eps']);
267
268 plot(diffs3(1,2:end),diffs3(2,2:end),'-')
269 xlabel('\bf iterations','fontsize',12);
270 ylabel('\bf normed difference to last step','fontsize',12);
271
272 filename = [figureFolder,'discrete_model_diff_ws_096'];
273 print('-dpng', [filename,'.png']);
274 print('-depsc2', [filename,'.eps']);
275
276
277 %% OUTPUT 3a - Opinion development graphs - Random
278 clear all;
279 close all;
280 clc;
281
282 % Set parameters
283 n = 3200;
284 k = 4;
285 gamma = 10;
286 iter = 0;
287 average_iterations = 1;
288 max_iterations = 20000000;
289 threshold = 0;
290 figureFolder = '../documentation/figures/';
291
292 fprintf('Random\n');
293
294
295 % Generate values
296 phi = 0.04;
297 d_it = 10^5;
298 fprintf('Part 1: phi = %1.3f\n', phi);
299 [~,ops1,~] = runModel(n,'random',k,gamma,phi,iter,average_iterations,...
300                     max_iterations,threshold,d_it);
301 phi = 0.458;
302 d_it = 1000;
303 fprintf('Part 2: phi = %1.3f\n', phi);
304 [~,ops2,~] = runModel(n,'random',k,gamma,phi,iter,average_iterations,...
305                     max_iterations,threshold,d_it);
306 phi = 0.96;
307 d_it = 1000;
308 fprintf('Part 3: phi = %1.3f\n', phi);
309 [~,ops3,~] = runModel(n,'random',k,gamma,phi,iter,average_iterations,...
310                     max_iterations,threshold,d_it);
311
312 % Plot graphic & save files
313 pcolor(ops1);
314 shading flat;
315 xlabel('\bf x10^5 iterations','fontsize',12);
316 ylabel('\bf opinions','fontsize',12);
317 set(gca,'YTickLabel',{})
318
319 filename = [figureFolder,'discrete_opinion_development_random_004'];
320 print('-dpng', [filename,'.png']);
321 print('-depsc2', [filename,'.eps']);

```

```

322
323
324
325 pcolor(ops2);
326 shading flat;
327 xlabel('\bf x10^3 iterations'),'fontsize',12);
328 ylabel('\bf opinions'),'fontsize',12);
329 set(gca,'YTickLabel',{})
330
331 filename = [figureFolder,'discrete_opinion_development_random_0458'];
332 print('-dpng', [filename,'.png']);
333 print('-depsc2', [filename,'.eps']);
334
335
336
337 pcolor(ops3);
338 shading flat;
339 xlabel('\bf x10^3 iterations'),'fontsize',12);
340 ylabel('\bf opinions'),'fontsize',12);
341 set(gca,'YTickLabel',{})
342
343 filename = [figureFolder,'discrete_opinion_development_random_096'];
344 print('-dpng', [filename,'.png']);
345 print('-depsc2', [filename,'.eps']);
346
347
348 surf(ops1);
349 shading interp;
350 axis tight
351 xlabel('\bf x10^5 iterations'),'fontsize',12);
352 ylabel('\bf opinions'),'fontsize',12);
353 zlabel('\bf group sizes'),'fontsize',12);
354 set(gca,'YTickLabel',{})
355
356 filename = [figureFolder,'discrete_opinion_development_random_3d_004'];
357 print('-dpng', [filename,'.png']);
358 print('-depsc2', [filename,'.eps']);
359
360
361 surf(ops2);
362 shading interp;
363 axis tight
364 xlabel('\bf x10^3 iterations'),'fontsize',12);
365 ylabel('\bf opinions'),'fontsize',12);
366 zlabel('\bf group sizes'),'fontsize',12);
367 set(gca,'YTickLabel',{})
368
369 filename = [figureFolder,'discrete_opinion_development_random_3d_0458'];
370 print('-dpng', [filename,'.png']);
371 print('-depsc2', [filename,'.eps']);
372
373
374 surf(ops3);
375 shading interp;
376 axis tight
377 xlabel('\bf x10^3 iterations'),'fontsize',12);
378 ylabel('\bf opinions'),'fontsize',12);
379 zlabel('\bf group sizes'),'fontsize',12);
380 set(gca,'YTickLabel',{})
381
382 filename = [figureFolder,'discrete_opinion_development_random_3d_096'];

```

```

383 print('-dpng', [filename, '.png']);
384 print('-depsc2', [filename, '.eps']);
385
386
387 %% OUTPUT 3b - Opinion development graphs - Small world
388
389 clear all;
390 close all;
391 clc;
392
393 % Set parameters
394 n = 3200;
395 k = 4;
396 gamma = 10;
397 iter = 0;
398 average_iterations = 1;
399 max_iterations = 20000000;
400 threshold = 0;
401 figureFolder = '../documentation/figures/';
402
403 fprintf('Small world\n');
404
405 % Generate values
406 phi = 0.04;
407 d_it = 10^5;
408 fprintf('Part 1: phi = %1.3f\n', phi);
409 [~,ops1,~] = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations,...
410                     max_iterations,threshold,d_it);
411 phi = 0.458;
412 d_it = 1000;
413 fprintf('Part 2: phi = %1.3f\n', phi);
414 [~,ops2,~] = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations,...
415                     max_iterations,threshold,d_it);
416 phi = 0.96;
417 d_it = 1000;
418 fprintf('Part 3: phi = %1.3f\n', phi);
419 [~,ops3,~] = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations,...
420                     max_iterations,threshold,d_it);
421
422 % Plot graphic & save files
423 pcolor(ops1);
424 shading flat;
425 xlabel('\bf x10^5 iterations','fontsize',12);
426 ylabel('\bf opinions','fontsize',12);
427 set(gca,'YTickLabel',{})
428 title('\bf Opinion distribution over time. $\phi = 0.04$'....
429       , 'interpreter','latex','fontsize',12);
430
431 filename = [figureFolder,'discrete_opinion_development_ws_004'];
432 print('-dpng', [filename, '.png']);
433 print('-depsc2', [filename, '.eps']);
434
435
436
437 pcolor(ops2);
438 shading flat;
439 xlabel('\bf x10^3 iterations','fontsize',12);
440 ylabel('\bf opinions','fontsize',12);
441 set(gca,'YTickLabel',{})
442 title('\bf Opinion distribution over time. $\phi = 0.458$'....
443       , 'interpreter','latex','fontsize',12);

```

```

444
445 filename = [figureFolder,'discrete_opinion_development_ws_0458'];
446 print('-dpng', [filename,'.png']);
447 print('-depsc2', [filename,'.eps']);
448
449
450
451 pcolor(ops3);
452 shading flat;
453 xlabel('\bf x10^3 iterations'),'fontsize',12);
454 ylabel('\bf opinions'),'fontsize',12);
455 set(gca,'YTickLabel',{})
456 title('\bf Opinion distribution over time.  $\phi = 0.96$ '...
457         , 'interpreter','latex','fontsize',12);
458
459 filename = [figureFolder,'discrete_opinion_development_ws_096'];
460 print('-dpng', [filename,'.png']);
461 print('-depsc2', [filename,'.eps']);
462
463
464 surf(ops1);
465 shading interp;
466 axis tight
467 xlabel('\bf x10^5 iterations'),'fontsize',12);
468 ylabel('\bf opinions'),'fontsize',12);
469 zlabel('\bf group sizes'),'fontsize',12);
470 set(gca,'YTickLabel',{})
471 title('\bf Opinion distribution over time.  $\phi = 0.04$ '...
472         , 'interpreter','latex','fontsize',12);
473
474 filename = [figureFolder,'discrete_opinion_development_ws_3d_004'];
475 print('-dpng', [filename,'.png']);
476 print('-depsc2', [filename,'.eps']);
477
478
479 surf(ops2);
480 shading interp;
481 axis tight
482 xlabel('\bf x10^3 iterations'),'fontsize',12);
483 ylabel('\bf opinions'),'fontsize',12);
484 zlabel('\bf group sizes'),'fontsize',12);
485 set(gca,'YTickLabel',{})
486 title('\bf Opinion distribution over time.  $\phi = 0.458$ '...
487         , 'interpreter','latex','fontsize',12);
488
489 filename = [figureFolder,'discrete_opinion_development_ws_3d_0458'];
490 print('-dpng', [filename,'.png']);
491 print('-depsc2', [filename,'.eps']);
492
493
494 surf(ops3);
495 shading interp;
496 axis tight
497 xlabel('\bf x10^3 iterations'),'fontsize',12);
498 ylabel('\bf opinions'),'fontsize',12);
499 zlabel('\bf group sizes'),'fontsize',12);
500 set(gca,'YTickLabel',{})
501 title('\bf Opinion distribution over time.  $\phi = 0.96$ '...
502         , 'interpreter','latex','fontsize',12);
503
504 filename = [figureFolder,'discrete_opinion_development_ws_3d_096'];

```

```

505 print('-dpng', [filename, '.png']);
506 print('-depsc2', [filename, '.eps']);
507
508
509 %% OUTPUT 4 - examine dependency on beta (parameter in Watts and Strogatz)
510 % Generated graphs examine the dependency on beta of the group size
511 % distribution. For that several graphs equivalent to the one of Figure 2
512 % in paper [1] are generated with different betas.
513
514 % reset
515 clear all;
516 close all;
517 clc;
518 if (matlabpool('size') ~= 0) % closes open workers
519     matlabpool close force;
520 end
521
522 % Set values
523 n = 3200;
524 k = 4;
525 gamma = 10;
526 average_iterations = 10;
527 iter = 20000000;
528 max_iterations = 500000;
529 threshold = 0.05;
530 d_it = 10000;
531
532 % The simulation is run for the following values (betas).
533 betas = 0:0.5:1;
534
535 % folder to save the figures
536 figureFolder = '../documentation/figures/';
537
538 % flag to turn the parallel toolbox on or off
539 parallel = false;
540
541 fprintf('Small world\n\n');
542
543 % Setup matlabpool workers to run job in parallel if demanded.
544 % (default configuration is used for matlabpool)
545 if parallel
546     matlabpool open;
547 end
548
549 for i = 1:size(betas,2)
550     fprintf('beta: %0.2f\n', betas(i));
551
552     % running simulation for several phi's
553     phi = 0.04;
554     fprintf('Part 1: phi = %1.3f\n', phi)
555     tic; dist1(:,i) = runModel(n,'watts_strogatz',k,gamma,phi,iter,...
556         average_iterations,max_iterations,threshold,d_it,betas(i)); toc;
557
558     phi = 0.458;
559     fprintf('Part 2: phi = %1.3f\n', phi)
560     tic; dist2(:,i) = runModel(n,'watts_strogatz',k,gamma,phi,iter,...
561         average_iterations,max_iterations,threshold,d_it,betas(i)); toc;
562
563     phi = 0.96;
564     fprintf('Part 3: phi = %1.3f\n', phi)
565     tic; dist3(:,i) = runModel(n,'watts_strogatz',k,gamma,phi,iter,...

```

```

566         average_iterations,max_iterations,threshold,d_it,betas(i)); toc;
567
568 % plotting
569 subplot(3,1,1)
570
571 subplot('Position',[.1 .1+1.6/3 .8 .8/3])
572 loglog(1:n,dist1(:,i)./(n/gamma),'o')
573 xlim([0 3200])
574 set(gca,'XTickLabel',{})
575 set(gca,'YTickLabel',' |0.01|0.1| ')
576 ylabel('P(s)')
577 string = '$\textbf{(a)} \quad \phi = 0.04$';
578 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
579     'fontsize',14,'units','norm')
580
581 subplot('Position',[.1 .1+.8/3 .8 .8/3])
582 loglog(1:n,dist2(:,i)./(n/gamma),'o')
583 xlim([0 3200])
584 set(gca,'XTickLabel',{})
585 set(gca,'YTickLabel',' |0.01|0.1| ')
586 ylabel('P(s)')
587 string = '$\textbf{(b)} \quad \phi = 0.458$';
588 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
589     'fontsize',14,'units','norm')
590
591 subplot('Position',[.1 .1 .8 .8/3])
592 loglog(1:n,dist3(:,i)./(n/gamma),'o')
593 xlim([0 3200])
594 set(gca,'XTickLabel',{'1','10','100','1000'})
595 set(gca,'YTickLabel',' |0.01|0.1| ')
596 xlabel('s')
597 ylabel('P(s)')
598 string = '$\textbf{(c)} \quad \phi = 0.96$';
599 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
600     'fontsize',14,'units','norm')
601
602 % saving the figures in different formats
603 filename = [figureFolder,'discrete_model_ws_k4_beta_', sprintf('%.2f',betas(i))];
604 print('-dpng', [filename,'.png']);
605 print('-depsc2', [filename,'.eps']);
606 saveas(gcf, [filename, '.fig']);
607
608 fprintf('\n');
609
610 end
611
612 % saving all the variables used for plotting in a file
613 save([figureFolder,'discrete_model_ws_k4_beta_05.mat']);
614
615 % close matlabpool workers if job was run in parallel
616 if parallel
617     matlabpool close;
618 end
619
620 %% OUTPUT 5 - examine dependency on beta and speed of convergence
621 % Generated graphs examine the dependency on beta on the speed of
622 % convergence. For that the number of iterations require to fall below
623 % the specified threshold are calculated for several betas and all the
624 % three values for phi.
625
626 % reset

```



```

627 clear all;
628 close all;
629 clc;
630 if (matlabpool('size') ~= 0) % closes open workers
631     matlabpool close force;
632 end
633
634 % Set values
635 n = 3200;
636 k = 4;
637 gamma = 10;
638 average_iterations = 10;
639 iter = 0;
640 max_iterations = 20000000;
641 threshold = 0.02;
642 d_it = 500;
643
644 % The simulation is run for the following values (betas).
645 betas = 0.0:0.5:1;
646
647 % folder to save the figures
648 figureFolder = '../documentation/figures/';
649
650
651 % flag to turn the parallel toolbox on or off
652 parallel = true;
653
654 fprintf('Small world\n\n');
655
656 % Setup matlabpool workers to run job in parallel if demanded.
657 % (default configuration is used for matlabpool)
658 if parallel
659     matlabpool open;
660 end
661
662 iterations_1 = zeros(size(betas,2),1);
663 iterations_2 = zeros(size(betas,2),1);
664 iterations_3 = zeros(size(betas,2),1);
665
666
667 parfor i = 1:size(betas,2)
668
669     fprintf('beta: %0.2f\n', betas(i));
670
671     phi = 0.04;
672     fprintf('Part 1: phi = %1.3f\n', phi)
673     tic; [~,~,iterations_1(i),~] = runModel(n,'watts_strogatz',k,gamma,phi,...
674         iter,average_iterations,max_iterations,threshold,d_it,betas(i)); toc;
675     phi = 0.458;
676     fprintf('Part 2: phi = %1.3f\n', phi)
677     tic; [~,~,iterations_2(i),~] = runModel(n,'watts_strogatz',k,gamma,phi,...
678         iter,average_iterations,max_iterations,threshold,d_it,betas(i)); toc;
679     phi = 0.96;
680     fprintf('Part 3: phi = %1.3f\n', phi)
681     tic; [~,~,iterations_3(i),~] = runModel(n,'watts_strogatz',k,gamma,phi,...
682         iter,average_iterations,max_iterations,threshold,d_it,betas(i)); toc;
683
684     fprintf('\n');
685 end
686
687 % plotting

```

```

688 plot(betas, [iterations_1, iterations_2, iterations_3]);
689 xlabel('\bf \beta', 'fontsize', 12);
690 ylabel('\bf number of iterations', 'fontsize', 12);
691 legend('\phi = 0.04', '\phi = 0.458', '\phi = 0.96');
692
693 % saving the figures in different formats
694 filename = [figureFolder, 'discrete_model_ws_k4_number_iterations'];
695 print('-dpng', [filename, '.png']);
696 print('-depsc2', [filename, '.eps']);
697 saveas(gcf, [filename, '.fig']);
698
699 % saving all the variables used for plotting in a file
700 save([figureFolder, 'discrete_model_ws_k4_number_iterations_all.mat']);
701
702 % close matlabpool workers if job was run in parallel
703 if parallel
704     matlabpool close
705 end
706
707 %% OUTPUT 6 - examine dependency on beta and speed of convergence
708 % Generated graphs examine the dependency on beta on the speed of
709 % convergence. For that the number of iterations require to fall below
710 % the specified threshold are calculated for several betas and phi = 0.04
711
712 % reset
713 clear all;
714 close all;
715 clc;
716 if (matlabpool('size') ~= 0) % closes open workers
717     matlabpool close force;
718 end
719
720 % Set values
721 n = 3200;
722 k = 4;
723 gamma = 10;
724 average_iterations = 1;
725 iter = 0;
726 max_iterations = 20000000;
727 threshold = 0.01;
728 d_it = 500;
729
730 % The simulation is run for the following values (betas).
731 betas = linspace(0,1,16);
732
733 % folder to save the figures
734 figureFolder = '../documentation/figures/';
735
736
737 % flag to turn the parallel toolbox on or off
738 parallel = true;
739
740 fprintf('Small world\n\n');
741
742 % Setup matlabpool workers to run job in parallel if demanded.
743 % (default configuration is used for matlabpool)
744 if parallel
745     matlabpool open;
746 end
747
748 iterations = zeros(size(betas,2),1);

```

```

749
750 parfor i = 1:size(betas,2)
751
752     fprintf('beta: %0.2f\n', betas(i));
753
754     phi = 0.04;
755     % fprintf('phi = %1.3f\n', phi)
756     tic; [~,~,iterations(i),~] = runModel(n,'watts_strogatz',k,gamma,phi,...
757         iter,average_iterations,max_iterations,threshold,d_it,betas(i)); toc;
758
759     fprintf('\n');
760 end
761
762 % plotting
763 plot(betas,iterations);
764 xlabel('\bf \beta','fontsize',12);
765 ylabel('\bf number of iterations','fontsize',12);
766 legend('\phi = 0.04');
767
768 % saving the figures in different formats
769 filename = [figureFolder,'discrete_model_ws_k4_number_iterations'];
770 print('-dpng', [filename,'.png']);
771 print('-depsc2', [filename,'.eps']);
772 saveas(gcf, [filename, '.fig']);
773
774 % saving all the variables used for plotting in a file
775 save([figureFolder,'discrete_model_ws_k4_number_iterations_all.mat']);
776
777 % close matlabpool workers if job was run in parallel
778 if parallel
779     matlabpool close
780 end
781
782 %% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
783 % GRAPHS FOR THE CONTINUOUS MODEL
784
785 %% OUTPUT 1
786 % Equivalent to Fig 1 in paper 2
787
788 % reset
789 clear all; close all; clc;
790 figure_handle = figure;
791
792 % setup the combined model to simulate the continuous model
793 graph = 'complete';
794 k = 0;
795 phi = 0;
796 std = 0;
797
798 % set parameters according to paper 2
799 n=3200;
800 % Paper: n = 500000;
801 % This requires to much space as the combined model allocates space
802 % for a complete graph which is n^2 space
803 u = 0.35;
804 mu = 0.001;
805 clusters = 1000; % u_0 = 0.001, clusters = 1/u_0
806 iterations = 400*n;
807 cskip = 400;
808
809 iterations = 80000; mu = 0.3; clusters = 100; cskip = 300;

```

```

810
811 % simulate
812 [histogram] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations);
813
814 % graph
815 pcolor( histogram(1:end-1,:) );
816 shading flat;
817 title( 'Opinion distribution over time' );
818 ylabel( 'Opinion' );
819 xlabel( 't/N' );
820
821 % save results
822 figureFolder = '../documentation/figures/';
823 filename = [figureFolder,'continuous_model'];
824 print('-dpng', [filename,'.png']);
825 print('-depsc2', [filename,'.eps']);
826 saveas(figure_handle, [filename, '.fig']);
827
828 %% OUTPUT 2
829 % Examine the 1/2u behaviour
830
831 % reset
832 clear all; close all; clc;
833 parallel = false;
834
835 % setup the combined model to simulate the continuous model
836 graph = 'complete';
837 k = 0;
838 phi = 0;
839 std = 0;
840
841 % set parameters for the continuous model
842 iterations = 80000;
843 mu = 0.3;
844 clusters = 100;
845 cskip = 250;
846 n = 3200;
847
848 % define the expected opinion clusters depending on u;
849 expected_clusters = [
850     1/(2*0.3); % minimum u to achieve consensus according to paper 2
851     2;
852     5;
853     8;
854     15 ];
855 expected_clusters(:,2) = 1./( 2 * expected_clusters(:,1) );
856
857 % initialize parallel computation
858 if parallel
859     matlabpool open
860 end
861
862 parfor i = 1:size(expected_clusters,1)
863     % simulate
864     e = expected_clusters(i,1);
865     u = expected_clusters(i,2);
866     [histogram] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations);
867     % graph
868     figure_handle = figure;
869     pcolor( histogram(1:end-1,:) );
870     shading flat;

```

```

871     title( sprintf('Opinion distribution over time. u = %2.2f',u) );
872     ylabel( 'Opinion' );
873     xlabel( 't/N' );
874
875     % save results
876     figureFolder = '../documentation/figures/';
877     filename = [figureFolder, sprintf('continuous_model_u_%2.2f',e) ];
878     print('-dpng', [filename, '.png']);
879     print('-depsc2', [filename, '.eps']);
880     saveas(figure_handle, [filename, '.fig']);
881 end
882
883 % cleanup
884 if parallel
885     matlabpool close
886 end
887
888 %% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
889 % GRAPHS FOR THE COMBINED MODEL
890
891 %% OUTPUT 1
892 % Generate a figure equivalent to the discrete model.
893
894 % reset
895 clear all; close all; clc;
896 parallel = false;
897 figure_handle = figure;
898
899 % configure the model
900 iterations = 1000000;
901 average_iterations = 10;
902 n = 3200;
903 k = 4;
904 clusters = 320;
905 cskip = 100;
906 graph = 'random';
907 Phi = [ 0.04, 0.458, 0.96];
908 std = 0.1;
909 u = 0.3;
910 mu = 0.3;
911
912 % Generate values
913 dist = zeros(n,average_iterations);
914
915 phi = Phi(1);
916 fprintf('Part 1: phi = %1.3f\n', phi);
917 for i=1:average_iterations
918     fprintf('Average %i/%i\n', i, average_iterations);
919     [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations);
920 end
921 dist1 = sum(dist,2)./average_iterations;
922
923 phi = Phi(2);
924 fprintf('Part 2: phi = %1.3f\n', phi);
925 for i=1:average_iterations
926     fprintf('Average %i/%i\n', i, average_iterations);
927     [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations);
928 end
929 dist2 = sum(dist,2)./average_iterations;
930
931 phi = Phi(3);

```

```

932 fprintf('Part 3: phi = %1.3f\n', phi);
933 for i=1:average_iterations
934     fprintf('Average %i/%i\n',i,average_iterations);
935     [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations);
936 end
937 dist3 = sum(dist,2)./average_iterations;
938
939 % Plot graphic
940 subplot('Position',[.1 .1+1.6/3 .8 .8/3])
941 loglog(1:n,dist1./(clusters),'o')
942 xlim([0 3200])
943 ylim([10^(-5.5) 0.2])
944 set(gca,'XTickLabel',{})
945 ylabel('P(s)')
946 string = '$\textbf{(a)} \quad \phi = 0.04$';
947 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
948     'fontsize',14,'units','norm')
949
950 subplot('Position',[.1 .1+.8/3 .8 .8/3])
951 loglog(1:n,dist2./(clusters),'o')
952 xlim([0 3200])
953 ylim([10^(-9) 0.2])
954 set(gca,'XTickLabel',{})
955 %set(gca,'YTickLabel',' |0.01|0.1| ')
956 ylabel('P(s)')
957 string = '$\textbf{(b)} \quad \phi = 0.458$';
958 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
959     'fontsize',14,'units','norm')
960
961 subplot('Position',[.1 .1 .8 .8/3])
962 loglog(1:n,dist3./(clusters),'o')
963 xlim([0 3200])
964 ylim([10^(-6) 0.2])
965 set(gca,'XTickLabel',{'1','10','100','1000'})
966 xlabel('s')
967 ylabel('P(s)')
968 string = '$\textbf{(c)} \quad \phi = 0.96$';
969 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
970     'fontsize',14,'units','norm')
971
972 % save results
973 figureFolder = '../documentation/figures/';
974 filename = [figureFolder, 'combined_model_random' ];
975 print('-dpng', [filename,'.png']);
976 print('-depsc2', [filename,'.eps']);
977 saveas(figure_handle, [filename, '.fig']);
978
979 %% OUTPUT 2
980 % Same as output 1, but for watts and strogatz graph beta=0.25
981
982 % reset
983 clear all; close all; clc;
984 parallel = false;
985 figure_handle = figure;
986
987 % configure the model
988 iterations = 1000000;
989 average_iterations = 10;
990 n = 3200;
991 k = 4;
992 clusters = 320;

```

```

993 cskip = 100;
994 graph = 'watts_strogatz';
995 Phi = [ 0.04, 0.458, 0.96];
996 std = 0.1;
997 u = 0.3;
998 mu = 0.3;
999
1000 beta = 0.25;
1001
1002 % Generate values
1003 dist = zeros(n,average_iterations);
1004
1005 phi = Phi(1);
1006 fprintf('Part 1: phi = %1.3f\n', phi);
1007 for i=1:average_iterations
1008     fprintf('Average %i/%i\n',i,average_iterations);
1009     [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
1010 end
1011 dist1 = sum(dist,2)./average_iterations;
1012
1013 phi = Phi(2);
1014 fprintf('Part 2: phi = %1.3f\n', phi);
1015 for i=1:average_iterations
1016     fprintf('Average %i/%i\n',i,average_iterations);
1017     [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
1018 end
1019 dist2 = sum(dist,2)./average_iterations;
1020
1021 phi = Phi(3);
1022 fprintf('Part 3: phi = %1.3f\n', phi);
1023 for i=1:average_iterations
1024     fprintf('Average %i/%i\n',i,average_iterations);
1025     [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
1026 end
1027 dist3 = sum(dist,2)./average_iterations;
1028
1029 % Plot graphic
1030 subplot('Position',[.1 .1+1.6/3 .8 .8/3])
1031 loglog(1:n,dist1./(clusters),'o')
1032 xlim([0 3200])
1033 ylim([10^(-5.5) 0.2])
1034 set(gca,'XTickLabel',{})
1035 ylabel('P(s)')
1036 string = '$\textbf{(a)} \quad \phi = 0.04$';
1037 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
1038     'fontsize',14,'units','norm')
1039
1040 subplot('Position',[.1 .1+.8/3 .8 .8/3])
1041 loglog(1:n,dist2./(clusters),'o')
1042 xlim([0 3200])
1043 ylim([10^(-9) 0.2])
1044 set(gca,'XTickLabel',{})
1045 %set(gca,'YTickLabel',' |0.01|0.1| ')
1046 ylabel('P(s)')
1047 string = '$\textbf{(b)} \quad \phi = 0.458$';
1048 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
1049     'fontsize',14,'units','norm')
1050
1051 subplot('Position',[.1 .1 .8 .8/3])
1052 loglog(1:n,dist3./(clusters),'o')
1053 xlim([0 3200])

```

```

1054 ylim([10^(-6) 0.2])
1055 set(gca,'XTickLabel',{'1','10','100','1000'})
1056 xlabel('s')
1057 ylabel('P(s)')
1058 string = '$\textbf{(c)} \quad \phi = 0.96$';
1059 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
1060     'fontsize',14,'units','norm')
1061
1062 % save results
1063 figureFolder = '../documentation/figures/';
1064 filename = [figureFolder, 'combined_model_watts_strogatz_beta_025' ];
1065 print('-dpng', [filename,'.png']);
1066 print('-depsc2', [filename,'.eps']);
1067 saveas(figure_handle, [filename, '.fig']);
1068
1069 %% OUTPUT 3
1070 % Same as output 1, but for watts and strogatz graph beta=0.5
1071
1072 % reset
1073 clear all; close all; clc;
1074 parallel = false;
1075 figure_handle = figure;
1076
1077 % configure the model
1078 iterations = 1000000;
1079 average_iterations = 10;
1080 n = 3200;
1081 k = 4;
1082 clusters = 320;
1083 cskip = 100;
1084 graph = 'watts_strogatz';
1085 Phi = [ 0.04, 0.458, 0.96];
1086 std = 0.1;
1087 u = 0.3;
1088 mu = 0.3;
1089
1090 beta = 0.5;
1091
1092 % Generate values
1093 dist = zeros(n,average_iterations);
1094
1095 phi = Phi(1);
1096 fprintf('Part 1: phi = %1.3f\n', phi);
1097 for i=1:average_iterations
1098     fprintf('Average %i/%i\n',i,average_iterations);
1099     [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
1100 end
1101 dist1 = sum(dist,2)./average_iterations;
1102
1103 phi = Phi(2);
1104 fprintf('Part 2: phi = %1.3f\n', phi);
1105 for i=1:average_iterations
1106     fprintf('Average %i/%i\n',i,average_iterations);
1107     [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
1108 end
1109 dist2 = sum(dist,2)./average_iterations;
1110
1111 phi = Phi(3);
1112 fprintf('Part 3: phi = %1.3f\n', phi);
1113 for i=1:average_iterations
1114     fprintf('Average %i/%i\n',i,average_iterations);

```



```

1115     [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
1116 end
1117 dist3 = sum(dist,2)./average_iterations;
1118
1119 % Plot graphic
1120 subplot('Position',[.1 .1+1.6/3 .8 .8/3])
1121 loglog(1:n,dist1./(clusters),'o')
1122 xlim([0 3200])
1123 ylim([10^(-5.5) 0.2])
1124 set(gca,'XTickLabel',{})
1125 ylabel('P(s)')
1126 string = '$\textbf{(a)} \quad \phi = 0.04$';
1127 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
1128     'fontsize',14,'units','norm')
1129
1130 subplot('Position',[.1 .1+.8/3 .8 .8/3])
1131 loglog(1:n,dist2./(clusters),'o')
1132 xlim([0 3200])
1133 ylim([10^(-9) 0.2])
1134 set(gca,'XTickLabel',{})
1135 %set(gca,'YTickLabel',' |0.01|0.1| ')
1136 ylabel('P(s)')
1137 string = '$\textbf{(b)} \quad \phi = 0.458$';
1138 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
1139     'fontsize',14,'units','norm')
1140
1141 subplot('Position',[.1 .1 .8 .8/3])
1142 loglog(1:n,dist3./(clusters),'o')
1143 xlim([0 3200])
1144 ylim([10^(-6) 0.2])
1145 set(gca,'XTickLabel',{'1','10','100','1000'})
1146 xlabel('s')
1147 ylabel('P(s)')
1148 string = '$\textbf{(c)} \quad \phi = 0.96$';
1149 text('string',string,'position',[0.7 0.8],'interpreter','latex',...
1150     'fontsize',14,'units','norm')
1151
1152 % save results
1153 figureFolder = '../documentation/figures/';
1154 filename = [figureFolder, 'combined_model_watts_strogatz_beta_050' ];
1155 print('-dpng', [filename, '.png']);
1156 print('-depsc2', [filename, '.eps']);
1157 saveas(figure_handle, [filename, '.fig']);
1158
1159
1160 %% OUTPUT 4
1161 % Examine the 1/2u behaviour in the combined model for a random graph
1162
1163 % reset
1164 clear all; close all; clc;
1165
1166 % configure the model
1167 iterations = 1000000;
1168 average_iterations = 10;
1169 n = 3200;
1170 k = 4;
1171 clusters = 320;
1172 cskip = 100;
1173 graph = 'random';
1174 Phi = [ 0.04, 0.458, 0.96];
1175 std = 0.1;

```

```

1176 u = 0.3;
1177 mu = 0.3;
1178
1179 % define the expected opinion clusters depending on u;
1180 expected_clusters = [
1181     1/(2*0.3); % minimum u to achieve consensus according to paper 2
1182     2;
1183     5;
1184     8;
1185     15 ];
1186 expected_clusters(:,2) = 1./( 2 * expected_clusters(:,1) );
1187
1188 for avg=1:average_iterations
1189
1190 for phi=Phi
1191
1192 for i = 1:size(expected_clusters,1)
1193     % simulate
1194     e = expected_clusters(i,1);
1195     u = expected_clusters(i,2);
1196     fprintf('Iteration %i/%i, phi=%1.3f, e=%2.2f\n',avg,average_iterations,phi,e);
1197     [histogram] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations);
1198     % graph
1199     figure_handle = figure;
1200     pcolor( histogram(1:end-1,:) );
1201     shading flat;
1202     title( sprintf('Opinion distribution over time. u = %2.2f',u) );
1203     ylabel( 'Opinion' );
1204     xlabel( 't/N' );
1205
1206     % save results
1207     figureFolder = './documentation/figures/';
1208     filename = [figureFolder, sprintf('combined_model_random_u_%2.2f_phi_%1.3f_i_%i',e,phi,avg)];
1209     print('-dpng', [filename, '.png']);
1210     print('-depsc2', [filename, '.eps']);
1211     saveas(figure_handle, [filename, '.fig']);
1212     close(figure_handle);
1213 end
1214
1215 end
1216
1217 end
1218
1219 %% OUTPUT 5
1220 % Examine the 1/2u behaviour in the combined model for
1221 % watts strogatz, beta = 0.25
1222
1223 % reset
1224 clear all; close all; clc;
1225
1226 % configure the model
1227 iterations = 1000000;
1228 average_iterations = 10;
1229 n = 3200;
1230 k = 4;
1231 clusters = 320;
1232 cskip = 100;
1233 graph = 'watts_strogatz';
1234 Phi = [ 0.04, 0.458, 0.96];
1235 std = 0.1;
1236 u = 0.3;

```

```

1237 mu = 0.3;
1238
1239 beta = 0.25;
1240
1241 % define the expected opinion clusters depending on u;
1242 expected_clusters = [
1243     1/(2*0.3); % minimum u to achieve consensus according to paper 2
1244     2;
1245     5;
1246     8;
1247     15 ];
1248 expected_clusters(:,2) = 1./( 2 * expected_clusters(:,1) );
1249
1250 for avg=1:average_iterations
1251
1252     for phi=Phi
1253
1254         for i = 1:size(expected_clusters,1)
1255             % simulate
1256             e = expected_clusters(i,1);
1257             u = expected_clusters(i,2);
1258             fprintf('Iteration %i/%i, phi=%1.3f, e=%2.2f\n',avg,average_iterations,phi,e);
1259             [histogram] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
1260             % graph
1261             figure_handle = figure;
1262             pcolor( histogram(1:end-1,:) );
1263             shading flat;
1264             title( sprintf('Opinion distribution over time. u = %2.2f',u) );
1265             ylabel( 'Opinion' );
1266             xlabel( 't/N' );
1267
1268             % save results
1269             figureFolder = '../documentation/figures/';
1270             filename = [figureFolder, ...
1271                 sprintf('combined_model_watts_strogatz_025_u_%2.2f_phi_%1.3f_i_%i',e,phi,avg) ];
1272             print('-dpng', [filename, '.png']);
1273             print('-depsc2', [filename, '.eps']);
1274             saveas(figure_handle, [filename, '.fig']);
1275             close(figure_handle);
1276         end
1277     end
1278 end
1279
1280 end
1281
1282 %% OUTPUT 6
1283 % Examine the 1/2u behaviour in the combined model for
1284 % watts strogatz, beta = 0.5
1285
1286 % reset
1287 clear all; close all; clc;
1288
1289 % configure the model
1290 iterations = 1000000;
1291 average_iterations = 10;
1292 n = 3200;
1293 k = 4;
1294 clusters = 320;
1295 cskip = 100;
1296 graph = 'watts_strogatz';
1297 Phi = [ 0.04, 0.458, 0.96];

```

```

1298 std = 0.1;
1299 u = 0.3;
1300 mu = 0.3;
1301
1302 beta = 0.5;
1303
1304 % define the expected opinion clusters depending on u;
1305 expected_clusters = [
1306     1/(2*0.3); % minimum u to achieve consensus according to paper 2
1307     2;
1308     5;
1309     8;
1310     15 ];
1311 expected_clusters(:,2) = 1./( 2 * expected_clusters(:,1) );
1312
1313 for avg=1:average_iterations
1314
1315     for phi=Phi
1316
1317         for i = 1:size(expected_clusters,1)
1318             % simulate
1319             e = expected_clusters(i,1);
1320             u = expected_clusters(i,2);
1321             fprintf('Iteration %i/%i, phi=%1.3f, e=%2.2f\n',avg,average_iterations,phi,e);
1322             [histogram] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
1323             % graph
1324             figure_handle = figure;
1325             pcolor( histogram(1:end-1,:) );
1326             shading flat;
1327             title( sprintf('Opinion distribution over time. u = %2.2f',u) );
1328             ylabel( 'Opinion' );
1329             xlabel( 't/N' );
1330
1331             % save results
1332             figureFolder = '../documentation/figures/';
1333             filename = [figureFolder, ...
1334                 sprintf('combined_model_watts_strogatz_050_u_%2.2f_phi_%1.3f_i_%i',e,phi,avg) ];
1335             print('-dpng', [filename, '.png']);
1336             print('-depsc2', [filename, '.eps']);
1337             saveas(figure_handle, [filename, '.fig']);
1338             close(figure_handle);
1339         end
1340     end
1341 end
1342
1343 end

```

D.6. generateOpinions.m

Listing 13: generateOpinions.m

```

1 function opinions = generateOpinions( n, factor )
2 %generateOpinions
3 % Generate a random opinion vector of length n.
4 % Factor can be used to specify the proportion of opinions in relation to
5 % n, where number_of_opinions = n / factor.
6
7 if nargin < 2, factor = 10; end
8

```

```

9 % Calculate the requested amount of opinions
10 count = n/factor;
11
12 % Generate the pseudorandom opinions
13 opinions = zeros(n,1);
14
15 for i = 1:n
16     opinions(i) = randi(count);
17 end
18
19 end

```

D.7. model.m

Listing 14: model.m

```

1 function [people, opinions] = model(people,opinions,phi,iter)
2 %model
3 % Model our system with given input parameters.
4 %
5 % At each step we select one person randomly.
6 % Then we use METHOD 1 with probability phi or METHOD 2 otherwise,
7 % to change the opinion or neighbourhood of the selected person.
8 %
9 % METHOD 1
10 % Select at random one of the edges attached to the selected person
11 % and move the other end of that edge to a vertex chosen
12 % randomly from the set off all vertices having the same
13 % opinion as the selected person.
14 %
15 % METHOD 2
16 % Pick a random neighbour of the selected person and set the opinion
17 % of the person to that of the selected neighbour.
18 %
19 % INPUT
20 % people Adjacency matrix representing the system
21 %
22 % opinions Vector of the agents opinions
23 %
24 % phi Probability factor for model
25 %
26 % iter Number of iterations to perform
27 %
28 % OUTPUT
29 % people Adjecency matrix after the model has been run
30 %
31 % opinions Opinions after the model has been run
32
33
34 if nargin < 4, error('Insufficient input arguments.');
```

```

44     person = randi(n);
45
46
47     % Check if the person is still connected to someone
48     % Do nothing otherwise
49     deg = nnz(people(person,:));
50     if deg == 0, continue; end
51
52
53     % Find all neighbours connected to person
54     neighbours = find(people(person,:));
55
56     % And select a neighbour at random
57     j = randi(length(neighbours));
58     neighbour = neighbours(j);
59
60     if neighbour == person
61         continue;
62     end
63
64     % Chose method 1 or 2 with probability phi
65     if rand() <= phi
66         % METHOD 1
67
68         % Find all people with the same opinion as person
69         opinion_group = find(opinions == opinions(person));
70
71         % Select one of the people with the same opinion at random
72         j = randi(length(opinion_group));
73         new_neighbour = opinion_group(j);
74
75         % Connect person and new_neighbour
76         % Disconnect person and old_neighbour
77         if person ~= new_neighbour
78             people(person,new_neighbour) = 1;
79             people(new_neighbour,person) = 1;
80             people(person,neighbour) = 0;
81             people(neighbour,person) = 0;
82         end
83     else
84         % METHOD 2
85
86         % Adopt opinion of neighbour
87         opinions(person) = opinions(neighbour);
88     end
89 end
90
91 end

```

D.8. runContinuousModel.m

Listing 15: runContinuousModel.m

```

1 function [histogram groups_distribution] = ...
2     runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta)
3 %             Setup and Run the Combined Model
4 % INPUT
5 % n         number of people in the system
6 % graph     type of social graph to use. valid input:

```

```

7 %           'random', 'watts_strogatz', 'complete'
8 % k         average number of connections per person
9 % phi       probability factor for model
10 % u        opinion threshold
11 % mu       convergence parameter
12 % std      standard derivation to find new friends by opinion
13 % clusters number of opinion clusters to observe
14 % cskip    skip factor for the returned histogram with respect to
15 %          the number of iterations
16 % iterations number of iterations
17 % beta     watts strogatz parameter
18 % OUTPUT
19 % histogram histogram according to the continuousModel function
20 % groups_distribution
21 %          vector where the i-th component is the amount of
22 %          of opinion groups of that size in the simulated system
23
24 % global DEBUG;
25
26 % input cosmetics
27 if nargin < 1, warning('Using defaults!'), n = 3200; end
28 if nargin < 2, graph = 'watts_strogatz'; end
29 if nargin < 3, k = 2; end
30 if nargin < 4, phi = 0.4; end
31 if nargin < 5, u = 0.04; end
32 if nargin < 6, mu = 0.3; end
33 if nargin < 7, std = 0.1; end
34 if nargin < 8, clusters = 100; end
35 if nargin < 9, cskip = 300; end
36 if nargin < 10, iterations = 1000000; end
37 if nargin < 11, beta = 0.25; end
38
39 % graph and opinion initialization
40 if strcmp(graph,'random')
41     people = createRandomSocialGraph(n,k);
42 elseif strcmp(graph,'watts_strogatz')
43     people = createWattsAndStrogatzModel(n,k,beta);
44 elseif strcmp(graph,'complete')
45     people = ones(n,n);
46 else
47     error('Unknown Social Graph');
48 end
49 opinions = generateContinuousOpinions(n);
50
51 % run the simulation with the specified parameters
52 [people, opinions, histogram] = ...
53     continuousModel(people,opinions,phi,u,mu,std,iterations,clusters,cskip);
54
55 % Overview of group sizes (how many groups of each size exist)
56 % partition the continuous opinions into #clusters opinion groups,
57 % get the distribution
58 opinion_distribution = hist(histogram(1:end-1,end),clusters);
59 % evaluate how many groups have the same size
60 groups_distribution = hist(opinion_distribution,1:n)';
61
62 % generate graphs only if the caller doesn't use the data
63 if nargin > 0
64     return
65 end
66
67 % plot the change of opinions over time

```

```

68 subplot(2,2,1);
69 pcolor(histogram(1:end-1,:));
70 shading flat;
71 title('Opinion distribution over time');
72
73 % plot the normed change in #cskip iterations over time
74 subplot(2,2,3);
75 plot(histogram(end,:));
76 ylim([0 30]);
77 title('Normed change during the iterations');
78
79 % plot the opinion distribution at the end
80 subplot(2,2,2);
81 hist(opinions,clusters);
82 title('Opinion distribution at the end');
83
84 % group size distribution
85 subplot(2,2,4);
86 loglog(groups_distribution,'o');
87 title('Group size distribution');
88
89 % 3D plot of the histogram. This is plotted in a new window
90 set(0,'CurrentFigure',figure)
91 surf(histogram(1:end-1,:));
92 shading interp;
93 colormap('Hot');
94
95 end

```

D.9. runModel.m

Listing 16: runModel.m

```

1 function [mean_distribution opinion_dist mean_number_iterations step_differences]...
2         = runModel(n,graph,k,gamma,phi,...
3                   iterations, average_iterations,...
4                   max_iterations, threshold, d_it, beta)
5 %runModel
6 % Simulate a number of steps of the model with the supplied parameters
7 % to compute the opinion group size distribution, i.e. how many opinion
8 % groups of sizes 1,...,n exist at the end of the simulation.
9 %
10 % We optionally average over several iterations.
11 % If iterations=0 is supplied, the function will run the simulation in
12 % steps of 10000 iterations and terminate as soon as the normed difference
13 % between the last two distributions reaches a certain threshold.
14 % In this case the opinion distributions are also aggregated in between
15 % runs and returned in opinion_dist.
16 %
17 % INPUT
18 % n           Number of agents in the system
19 %
20 % graph       Use random network ('random')
21 %            or small world ('watts_strogatz')?
22 %
23 % k           Average number of connections per person
24 %
25 % gamma       Ratio of the number of opinions to the number of people
26 %            gamma = (number of people) / (number of opinions)

```



```

27 %
28 % phi          Probability factor with which to choose
29 %              the method to be used in each iteration
30 %
31 % iterations    Number of iterations to perform with the simulation
32 %              If 0, the function will terminate, when a certain
33 %              threshold for the normed difference between
34 %              computation steps is reached
35 %
36 % average_iterations Number of times the whole model will be run
37 %              The results will be averaged over these runs
38 % OPTIONAL
39 % max_iterations Maximum number of iterations to perform when
40 %              iterations=0
41 %
42 % threshold     Which normed difference between steps has to be
43 %              reached before stopping?
44 %
45 % d_it          Number of iterations performed between checks if
46 %              there are significant changes.
47 %
48 % beta          The parameter beta from the Watts and Strogatz
49 %              model.
50 %
51 % OUTPUT
52 %
53 % mean_distribution Averaged opinion group size distribution
54 %
55 % opinion_dist    Opinion distributions at every thousand iteration
56 %              steps. Only calculated if iterations=0.
57 %
58 % mean_number_iterations
59 %              Average number of iterations required to fall below
60 %              the given threshold.
61 %
62 % step_differences Matrix with the number of iterations performed so far
63 %              in the first row and the respective difference
64 %              since the last step (d_it iterations) in the second
65 %
66
67 if nargin < 7, error('Insufficient input arguments.');
```

```

end
68 if nargin < 8, max_iterations = 200000; end
69 if nargin < 9, threshold = 0.05; end
70 if nargin < 10, d_it = 10000; end
71 if nargin < 11, beta = 0.25; end
72
73 % If the number of iterations is not specified, we will try to guess
74 % when to stop iterating, by observing whether the model still
75 % changes significantly over several iterations (specified by d_it).
76 if iterations == 0
77
78     % Distributions for comparison
79     dist_old = zeros();
80     dist_new = zeros();
81
82     % Matrix to aggregate the opinion distributions
83     opinion_dist = zeros(n/gamma,1);
84
85     % Number of iterations required to fall below the threshold
86     number_iterations = zeros(average_iterations, 1);
87 else

```

```

88     number_iterations = iterations;
89     if nargout > 1, error('Too many output arguments.');
```

```
90 end
```

```
91
```

```
92
```

```
93 % Opinion group distributions for all iterations, will be averaged
```

```
94 distribution = zeros(average_iterations,n);
```

```
95
```

```
96
```

```
97
```

```
98 % Compute 'average_iterations' computations
```

```
99 for i=1:average_iterations
```

```
100     % Print progress (if more than one run)
```

```
101     if (average_iterations > 1)
```

```
102         fprintf('Progress %3.2f%%\n', ((i-1)/average_iterations)*100)
```

```
103     end
```

```
104
```

```
105     % Model initialization
```

```
106     if strcmp(graph,'random')
```

```
107         people = createRandomSocialGraph(n,k);
```

```
108     elseif strcmp(graph,'watts_strogatz')
```

```
109         people = createWattsAndStrogatzModel(n,k,beta);
```

```
110     else
```

```
111         error('Unknown graph structure');
```

```
112     end
```

```
113
```

```
114     opinions = generateOpinions(n,gamma);
```

```
115
```

```
116     if iterations ~= 0
```

```
117         % CONSTANT NUMBER OF ITERATIONS
```

```
118         % Run model for 'iterations' steps
```

```
119         [~, opinions] = model(people,opinions,phi,iterations);
```

```
120
```

```
121         % Compute the opinion distribution with the histogram function 'hist'
```

```
122         % There are (n/gamma) opinions, so we want to bin the values
```

```
123         % into (n/gamma) groups
```

```
124         % We pass a vector with the exact binning points to be used
```

```
125         opinion_distribution = hist(opinions,1:(n/gamma));
```

```
126
```

```
127         % Compute the opinion group size distribution
```

```
128         group_distribution = hist(opinion_distribution,1:n);
```

```
129     else
```

```
130         % VARIABLE NUMBER OF ITERATIONS
```

```
131         % Run the model in steps of 'd_it' iterations
```

```
132         % and check the difference after each step
```

```
133         for j=1:(max_iterations/d_it)
```

```
134             % Continue to run the simulation with the specified parameters
```

```
135             [people, opinions] = model(people,opinions,phi,d_it);
```

```
136
```

```
137
```

```
138         % Compute the opinion distribution with the histogram function 'hist'
```

```
139         % There are (n/gamma) opinions, so we want to bin the values
```

```
140         % into (n/gamma) groups
```

```
141         % We pass a vector with the exact binning points to be used
```

```
142         opinion_distribution = hist(opinions,1:(n/gamma));
```

```
143
```

```
144         % Add the momentary opinion distribution
```

```
145         % Only computed for one computation (i.e. the last averaging step)
```

```
146         if (i==average_iterations)
```

```
147             opinion_dist(:,j) = opinion_distribution';
```

```
148         end
```

```

149
150     % Compute the opinion group size distribution
151     group_distribution = hist(opinion_distribution,1:n);
152
153
154     % Compute the normed difference between the last steps
155     dist_old = dist_new;
156     dist_new = group_distribution;
157     diff = norm(dist_new-dist_old)/norm(dist_new);
158
159     % Add the normed difference between the last steps
160     % Only computed for one computation (i.e. the last averaging step)
161     if (i==average_iterations)
162         step_differences(1,j) = j*d_it;
163         step_differences(2,j) = diff;
164     end
165
166     % Abort if threshold is reached
167     if (diff <= threshold)
168         break;
169     end
170 end
171 fprintf('Iterations performed: %d\n',j*d_it);
172 number_iterations(i) = j*d_it;
173 end
174
175 % Add final group size distribution to distribution matrix
176 distribution(i,:) = group_distribution;
177
178 end
179
180 if (average_iterations > 1)
181     fprintf('Progress 100%%\n')
182 end
183
184 % Compute the mean values for the distribution and number of iterations
185 % if averaging requested
186 if average_iterations > 1
187     mean_distribution = mean(distribution);
188     %mean_number_iterations = mean(number_iterations);
189 else
190     mean_distribution = distribution;
191     %mean_number_iterations = number_iterations;
192 end
193 end

```

E. Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich diese Gruppenarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen-Hilsmittel verwenden habe, und alle Stellen, die wörtlich oder sinngemäss aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Gruppenarbeit nicht, auch nicht auszugsweise, bereits für andere Prüfung ausgefertigt wurde.

Janosch Hildebrand

Jeremia Bär

Raphael Fuchs

F. Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Janosch Hildebrand

Jeremia Bär

Raphael Fuchs