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

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 model[1] 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

[1] in 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 then 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

[2]Given 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 forumla $count(u) = \frac{1}{2u}$.

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 completly 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.

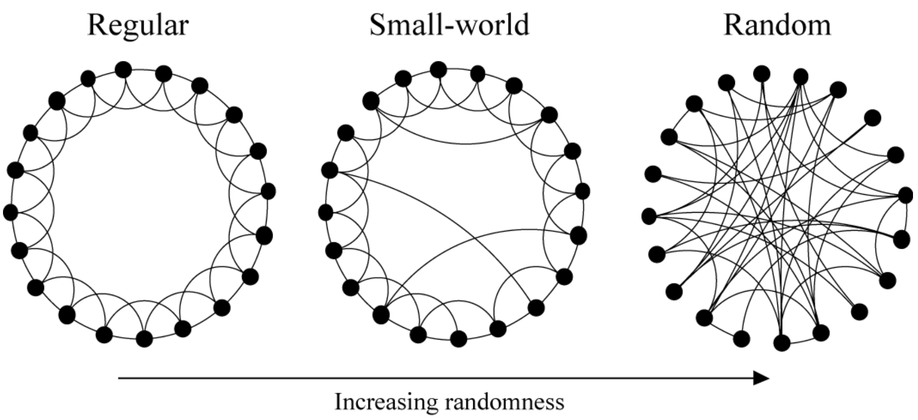


Figure 1: Different network structures for $n = 20$ and $k = 4$.

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)

[4] Version 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

```
% Generate initial graph structure and opinions
[...]

for j=1:(max_iterations/d_it)
  % Continue to run the simulation with the specified parameters
  [people, opinions] = model(people,opinions,phi,d_it);

  % Compute the opinion distribution with the histogram function 'hist'
  % There are (n/gamma) opinions, so we want to bin the values
  % into (n/gamma) groups
  % We pass a vector with the exact binning points to be used
  opinion_distribution = hist(opinions,1:(n/gamma));

  % Add the momentary opinion distribution
  % Only computed for one computation (i.e. the last averaging step)
  if (i==average_iterations)
    opinion_dist(:,j) = opinion_distribution';
  end

```

```
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 threshhold 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`

```
% create sparse nxn matrix A representing regular ring lattice (step 1)
[...]
A = sparse(row_index, col_index, ones(n*k, 1), n, n, n*k);

% For every node take every edge (n_i, n_j) with i < j and rewire it
% with probability beta. (step 2)
for i=1:n
    neighbours = find(A(i,:));
    for j=neighbours
        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 ploting 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`

```
% histogram stores the count of poeple holding the opinion in each cluster
% and the norm in clusterization change.
histogram = zeros(clusters + 1,floor(iter/cskip));

% in order to find people with a similiar opinion, a sorted copy
% of the opinions is created with a link to the original person
% index: sorted_opinions = [ opinion , index ]
sorted_opinions = [opinions , (1:n)'];
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
```

```
    diff_n1 = neighbours_opinion - sorted_opinions(i,1);
    diff_n2 = neighbours_opinion - sorted_opinions(i-1,1);
    if abs(diff_n1) < abs(diff_n2)
        new_neighbour = sorted_opinions(i,2);
    else
        new_neighbour = sorted_opinions(i-1,2);
    end
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 seperate 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 todays multicore computer systems. Instead we used the *Parallel Computing Toolbox* when running several simulations with varying paramters. 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.

[5] For 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 *Method 1* or *Method 2* 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 – *Number of opinions* $= N/\gamma$
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 $\phi \to 1$ edges are moved to agents sharing opinions, thus disconnecting groups of different opinions. In contrast, for $\phi \to 0$ only opinions are changed and so the simulation only reaches consensus when all connected agents have equal opinions.

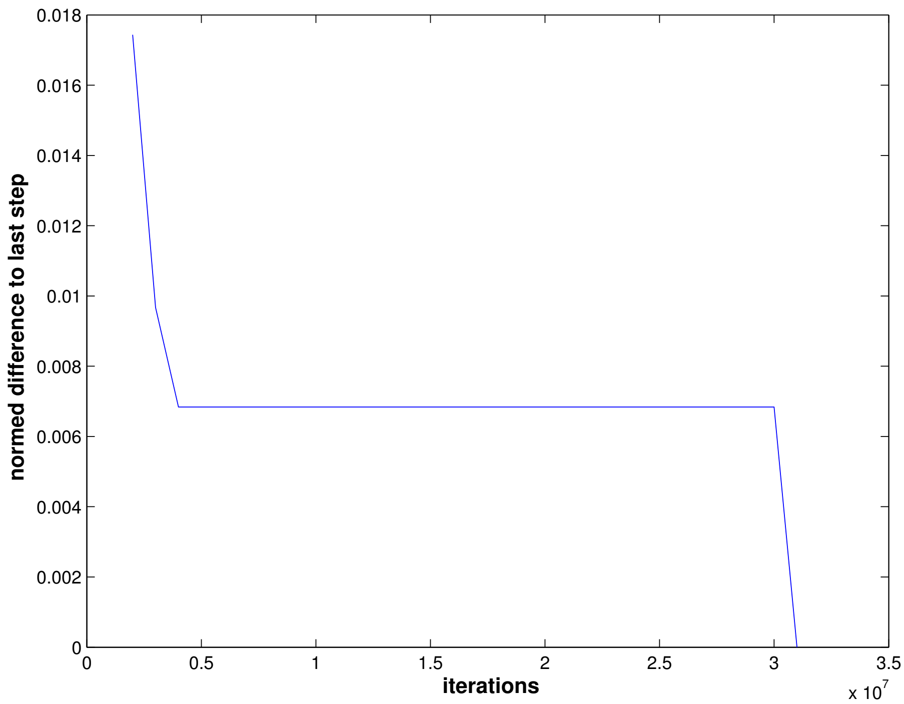


Figure 2: Convergence of model $\phi = 0.04$

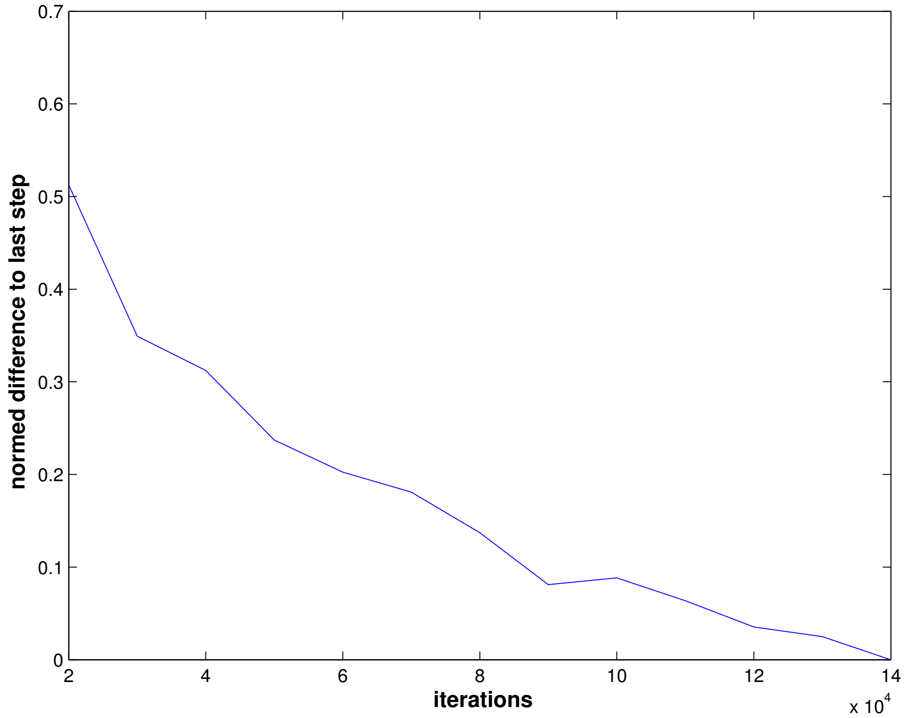


Figure 3: Convergence of model $\phi = 0.458$

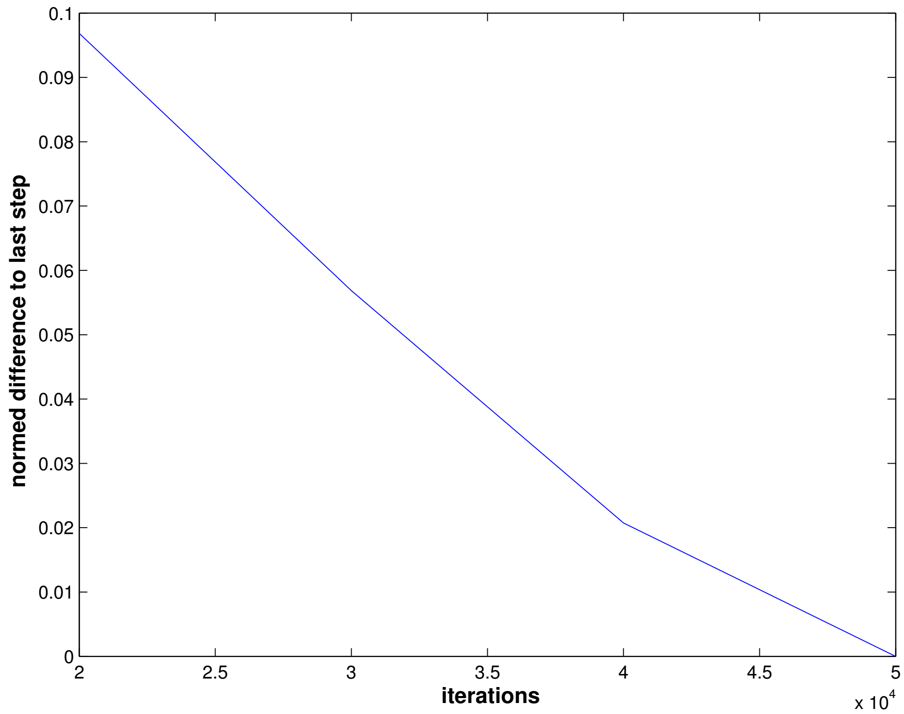


Figure 4: Convergence of model $\phi = 0.96$

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. Insted 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.

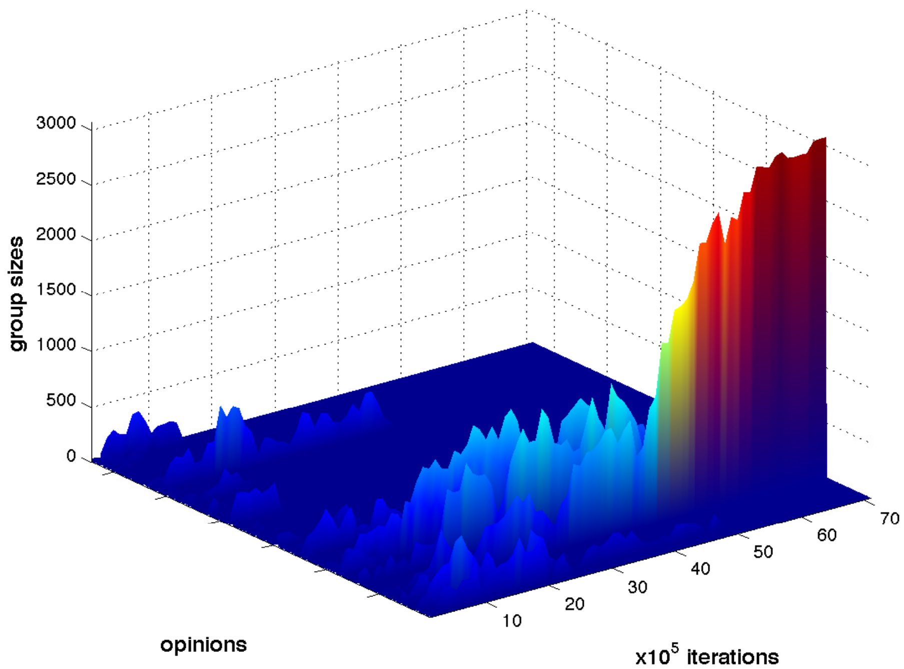


Figure 5: Opinion development over time for $\phi = 0.04$
We can see that as expected the opinions change intensively. At the beginning most opinion groups have sizes between 0 and approximately 100 with a mean of $\gamma = 10$. Very quickly several larger groups emerge. When the simulation is run long 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.

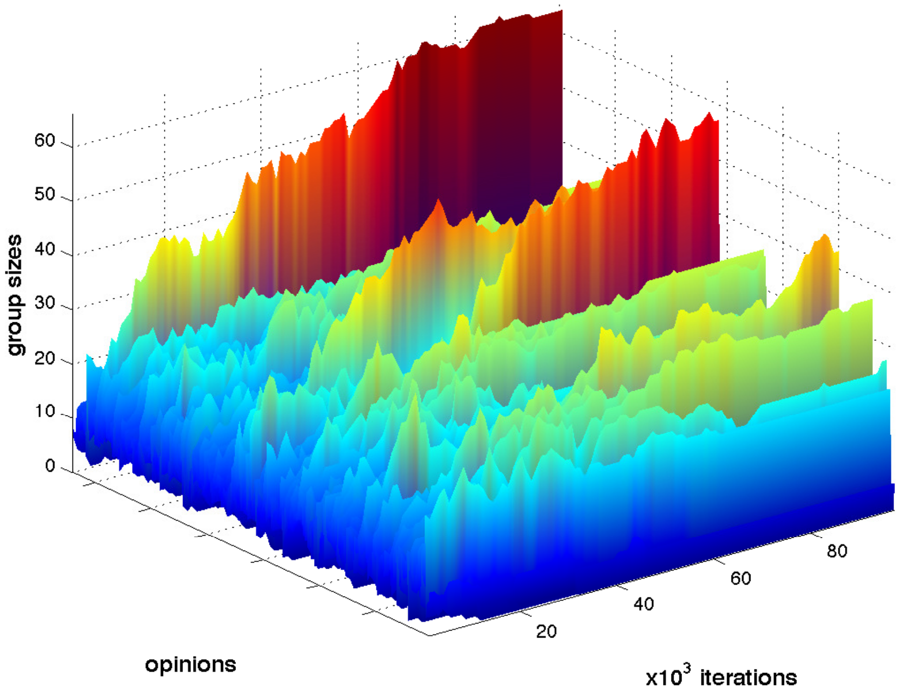


Figure 6: Opinion development over time for $\phi = 0.458$
As could be expected using the data in paper [1], this value of ϕ lets some opinion exchange occur but after some time the network degenerates into several medium-sized, disconnected groups. This is caused by the also equally probable destruction of connections in the network that connect agents of different opinions.

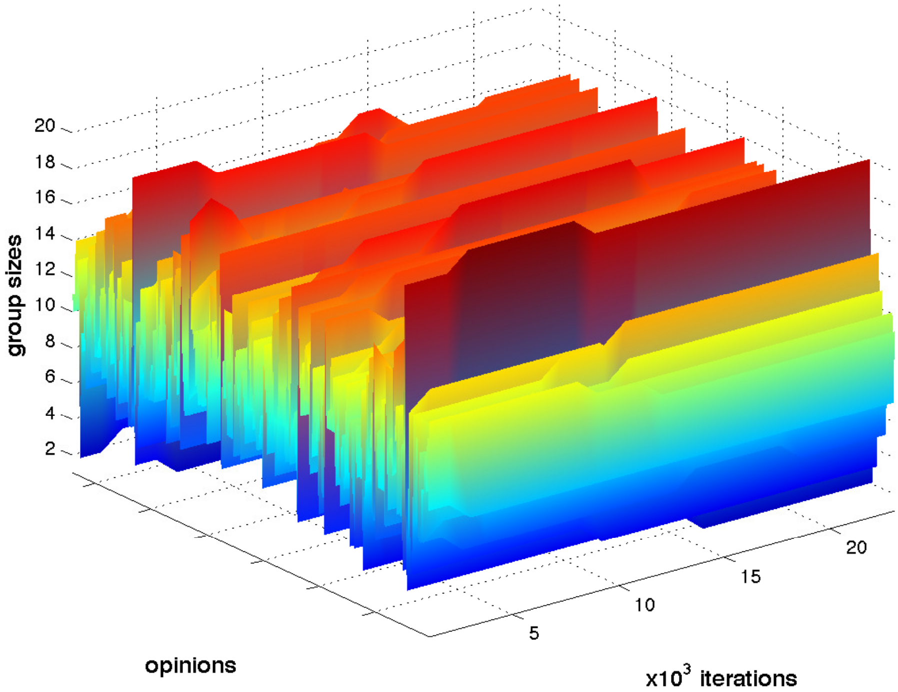


Figure 7: Opinion development over time for $\phi = 0.96$
Since for such a large ϕ the model mostly connects agents of equal opinions, the actual number of people having a certain opinion stays almost completely unchanged over the course of the simulation.

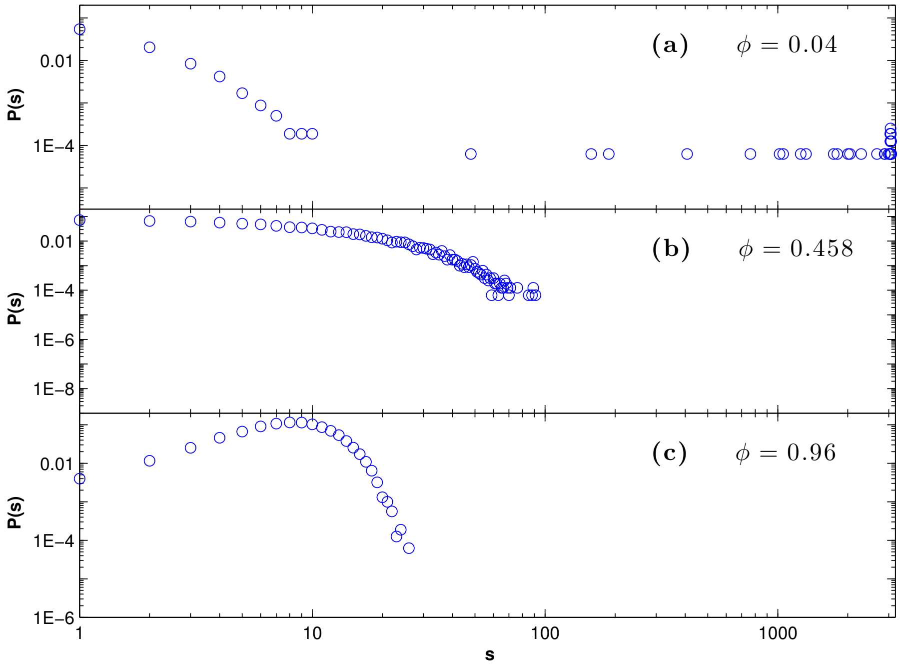


Figure 8: Equivalent to *Figure 2* from paper [1] with our discrete model used for simulation. The figure represents histograms of the community sizes s in the consensus state. The x-axis represents the size of opinion groups, whereas the y-axis specifies proportion of groups of a specific size compared 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) $\phi =$ **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) $\phi =$ **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) $\boldsymbol{\phi = 0.96}$

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 $\beta \to 0.0$ or a random graph $\beta \to 1.0$. 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.

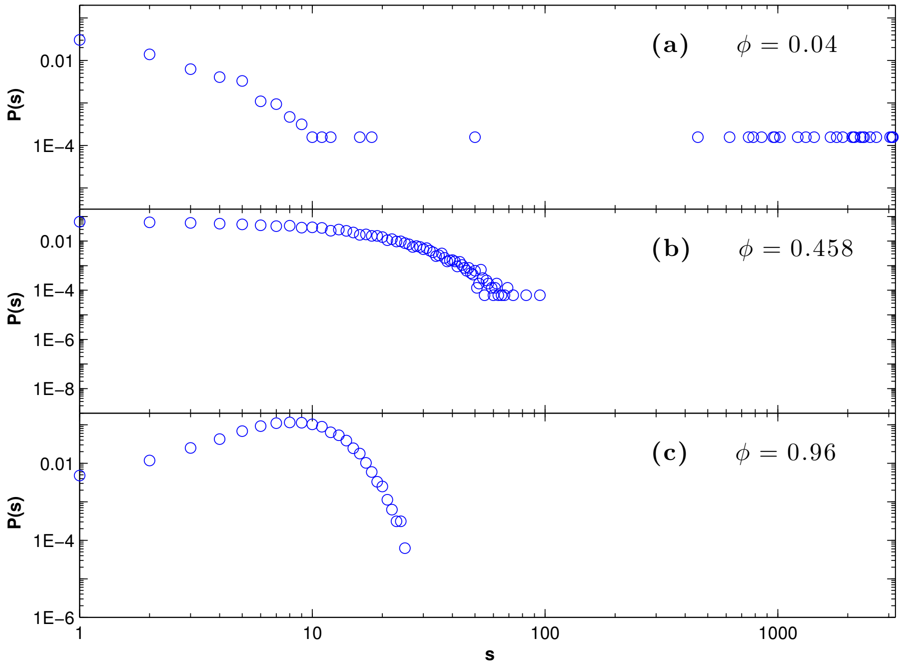


Figure 9: Equivalent to *Figure 2* from paper [1]. Computed using our discrete model with a Watts Strogatz network.
See Figure 8 for more information and parameter values. The value of β was set to 0.25.

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.

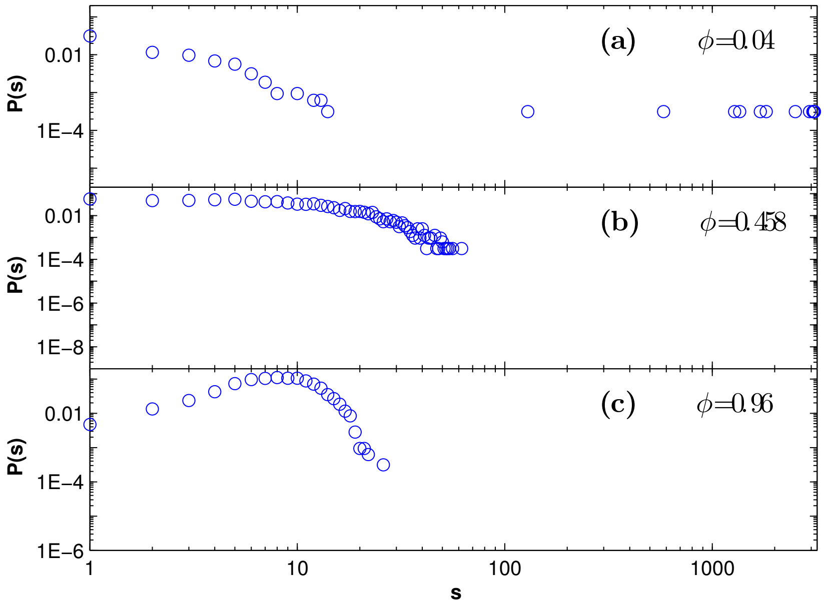


Figure 10: Opinion group size distribution with Watts and Strogatz model and $\beta = 0.00$

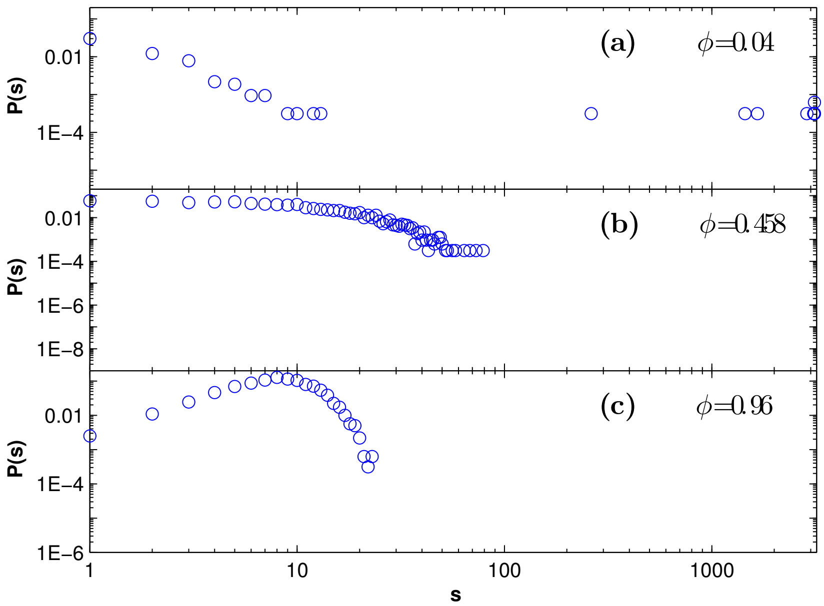


Figure 11: Opinion group size distribution with Watts and Strogatz model and $\beta = 0.50$

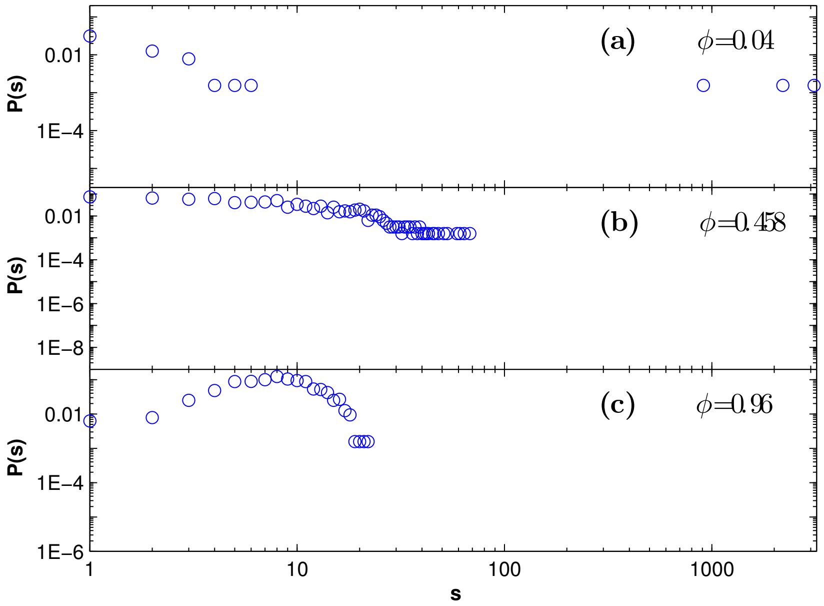


Figure 12: Opinion group size distribution with Watts and Strogatz model and $\beta = 1.00$

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 $\beta = 1.0$.

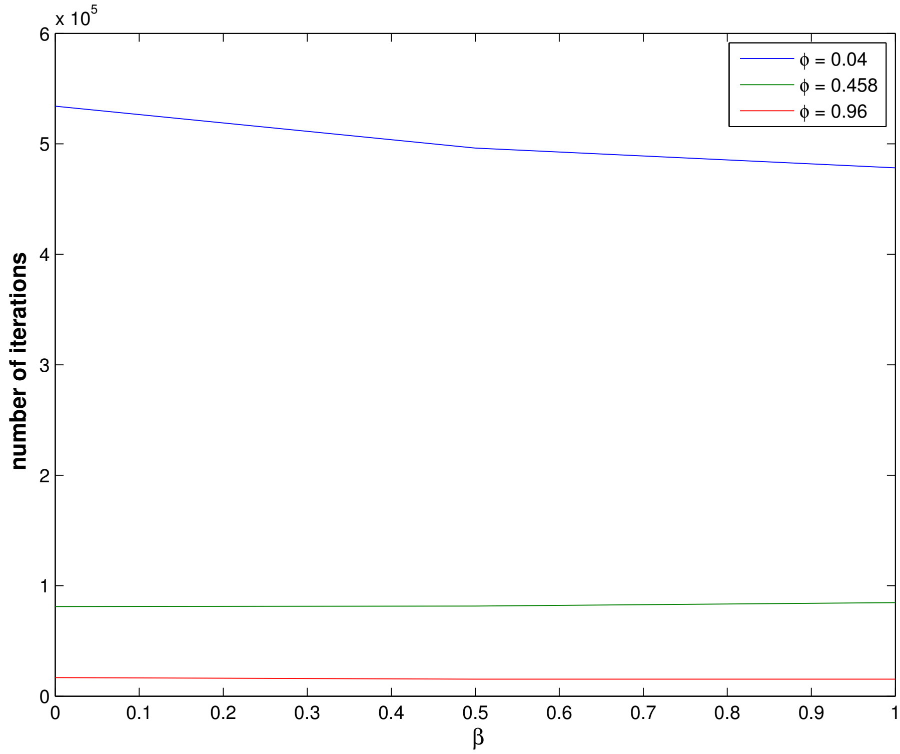


Figure 13: Number of iterations required to fall below threshold 0.05 for $\beta = 0.0$, $\beta = 0.5$ and $\beta = 1.0$

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 model[6]. The continuous, not network-based model presented in [2] was known the 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.

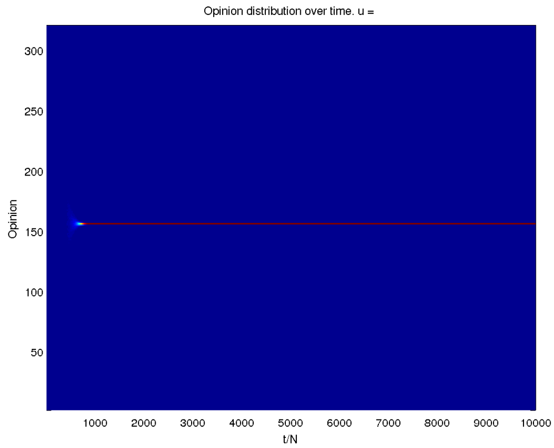
The varying settings are the underlaying graph and the parameter ϕ describing the probability of choosing *Method 1 or 2* of the agent interaction in each simulation step as already known from the discrete model. For ϕ we chose the values $0.040, 0.458$ and 0.960 as in the discrete model. To model the 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

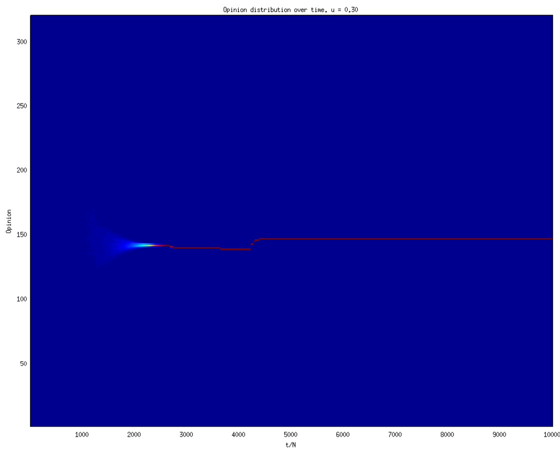
The incorporation of a network structure into the continous model leading to the combined model had a huge impact on the opinion formation in the system. Research question $2a$ was therefore revealed to be the most interesting one and is discussed in detail in this section. Research questions $2b$ and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

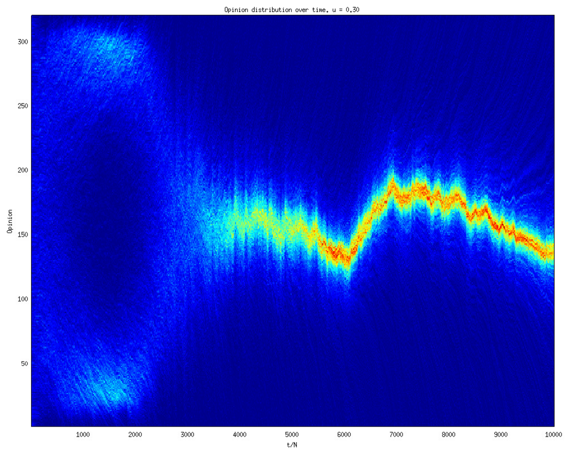
The combined model was found to form disjoint opinion groups with the same pattern $count(u) = \frac{1}{2u}$ as the continuous model. However the convergence differs hugely based on the chosen parameters. Furthermore, even though the opinion groups separate themselves from each other, they tend to not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



(a) Continuous Model



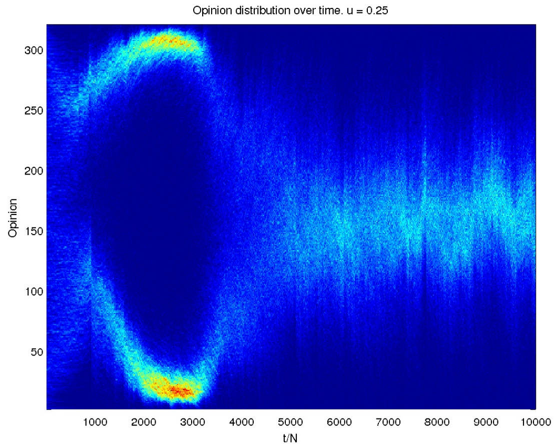
(b) Combined Model, $\phi = 0.040$



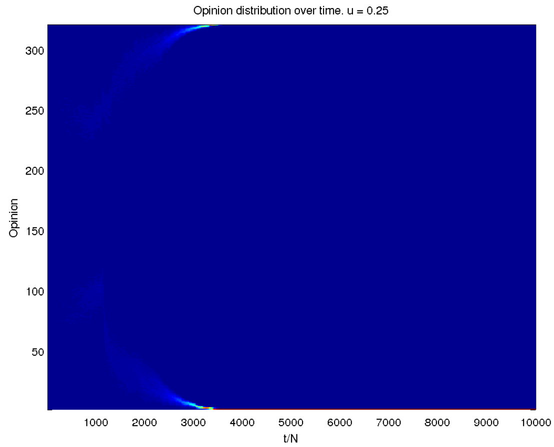
(c) Combined Model, $\phi = 0.458$

Figure 14: Convergence behaviour from continuous and combined model.

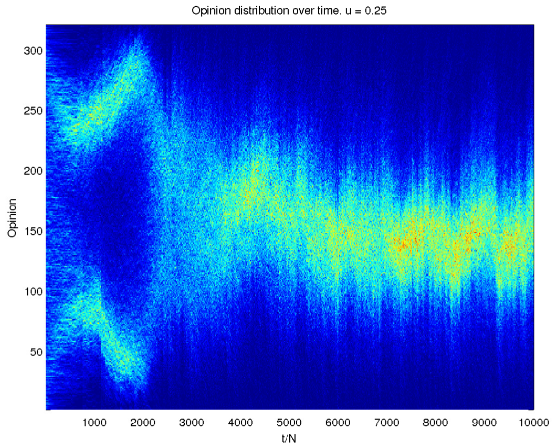
Note that for all graphs the same simulation settings were chosen. The count of iterations on the x-axis has to be multiplied by 100 as `cskip` = 100.
One can see that the continuous model in Figure 14a converges very fast in less than 100 000 iterations. The combined model in Figure 14b also has a very 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



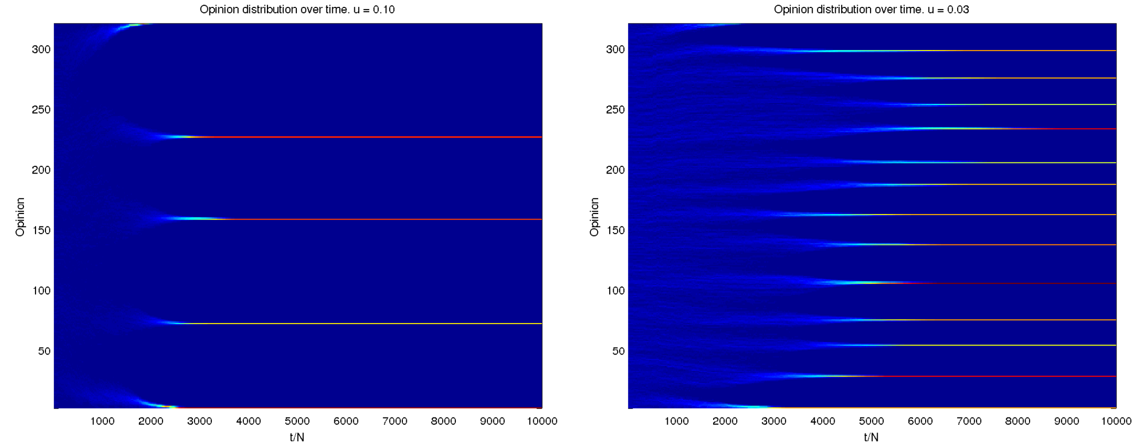
(b) Example of a clear decision



(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 seperate 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.

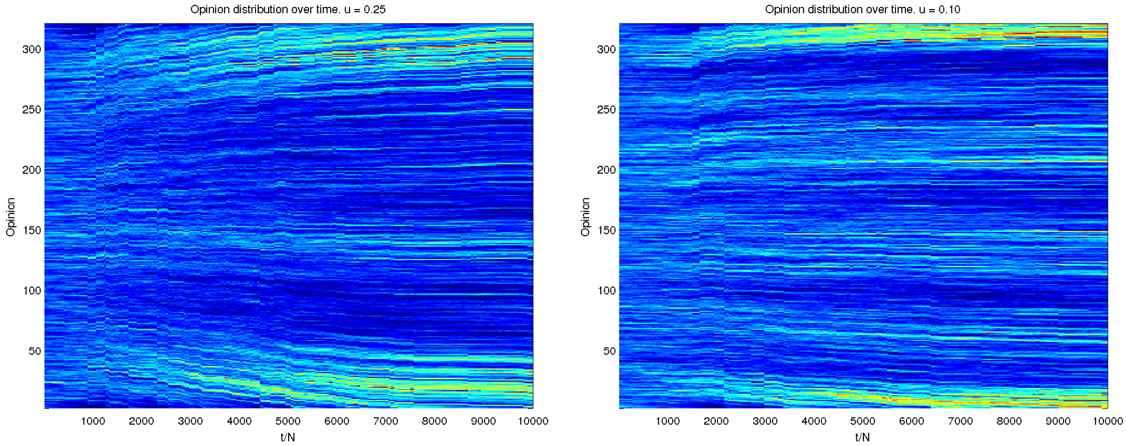


(a) Expected groups $\frac{1}{2u} \approx 5$ **(b)** Expected groups $\frac{1}{2u} \approx 15$

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 2*c*. 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 choosen 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.



(a) Expected groups $\frac{1}{2u} \approx 2$

(b) Expected groups, $\frac{1}{2u} \approx 5$

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 ontinuous values, further research into those areas is neccessary 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, primariliy running the simulation for the proposed 10 000 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 inital opinions are distributed randomly and do not depend on the local structure. The opinion developement 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 seperate 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 insuficciently 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

C. List of Tables

D. MATLAB®code

D.1. continuousModel.m

Listing 8: continuousModel.m

```
function [people, opinions, histogram] = ...
    continuousModel(people,opinions,phi,u,mu,std,iter,clusters,cskip)
%                           The combined Model.
% INPUT
%   people     adjacency matrix representing the system
%   opinions   vector of the opinions of the agents
%   phi        probability factor for model
%   u          opinion threshold
%   mu         convergence parameter
%   std        standard derivation used in probability to find
%              like-minded people
%   iter       number of iterations to perform
%   clusters   how many opinion clusters the histogram should contain (u0)
%   cskip      skip factor for the returned histogram with respect to
%              the number of iterations
% OUTPUT
%   people     adjacency matrix after the simulation
%   opinions   opinion vector after the simulation
%   histogram  this is a ( clusters+1 x floor(iter/skip) ) matrix.
%              the first #clusters rows contain the count of people
%              holding an opinion in that cluster. the j-th column
%              represents the state after (j-1)*cskip iterations.
%              the last row contains the norm of the change in the
%              clusterization process.

% Number of people in the system
n = length(opinions);

% histogram stores the count of poeple holding the opinion in each cluster
% and the norm in clusterization change.
histogram = zeros(clusters + 1,floor(iter/cskip));

% in order to find people with a similiar opinion, a sorted copy
% of the opinions is created with a link to the original person
% index: sorted_opinions = [ opinion , index ]
sorted_opinions = [opinions , (1:n)'];
sorted_opinions = sortrows(sorted_opinions);

old_hist = zeros(clusters,1);
% Main loop, iterating over t
for t=1:iter

    % print progress
    if exist('DEBUG')
        if mod(t,50000) == 0
            fprintf('%i/%i\n',t,iter);
        end
    end

    % update histogram
    if mod(t,cskip) == 0
        h = hist(opinions,clusters)';
        histogram(:,t/cskip) = [ h; norm(old_hist - h) ];
        old_hist = h;
```

```
    end

    % Select a person randomly
    person = randi(n);

    % Check if the person is still connected to someone
    % Do nothing otherwise
    deg = sum(people(person,:));
    if deg == 0, continue; end

    % Find all neighbours connected to that person
    neighbours = find(people(person,:));

    % Select a neighbour at random
    j = randi(length(neighbours));
    neighbour = neighbours(j);

    % Chose method 1 or 2 with probability phi
    if rand() <= phi
       % METHOD 1
       % Select at random one of the edges attached to person
       % and move the other end of that edge to a another person.
       % The new neighbour is choosen by a probability that is
       % normally distributed according to the difference in
       % opinion of the possibly new neighbour.

       % figure out what opinion the new neighbour should have
       neighbours_opinion = opinions(person) + std*randn();
       % do not accept opinions out of range [0,1]
       % try to find another person
       acceptable = 0;
       for i=1:100
           if neighbours_opinion < 0 || neighbours_opinion > 1
               neighbours_opinion = opinions(person) + std*randn();
           else
               acceptable = 1;
               break;
           end
       end
       % if that person has such an extreme opinion that it can't find
       % anyone within range [0,1], the extrema is taken
       if acceptable == 0
           if neighbours_opinion < 0
               neighbours_opinion = 0;
           elseif neighbours_opinion > 1
               neighbours_opinion = 1;
           end
       end;

       % find the person with the closest opinion to
       % neighbours_opinion
       i = find(sorted_opinions(:,1) > neighbours_opinion, 1 , 'first');
       if size(i,1) == 0
           % no one has such a strong (-> 1) opinion.
           % Take the person with the strongest opinion
           new_neighbour = sorted_opinions(end,2);
       elseif i == 1
           % there is no person with a lower opinion, so the person
           % at index i is the new neighbour
           new_neighbour = sorted_opinions(i,2);
       else
```

```
            % the person at index i-1 might be closer to the opinion
            diff_n1 = neighbours_opinion - sorted_opinions(i,1);
            diff_n2 = neighbours_opinion - sorted_opinions(i-1,1);
            if abs(diff_n1) < abs(diff_n2)
                new_neighbour = sorted_opinions(i,2);
            else
                new_neighbour = sorted_opinions(i-1,2);
            end
        end

        % Connect person and new_neighbour
        % Disconnect person and old_neighbour
        if person ~= new_neighbour
            people(person,new_neighbour) = 1;
            people(new_neighbour,person) = 1;
            people(person,neighbour) = 0;
            people(neighbour,person) = 0;
        end
    else
        % METHOD 2
        % Pick a random neighbour of person. If their opinion
        % differ by more than the opinion threshold u, do nothing.
        % Else, converge the two opinions according to the
        % convergence parameter mu.

        diff = opinions(neighbour) - opinions(person);
        if( abs( diff ) <= u )
            % update opinion vector
            opinions(person) = opinions(person) + mu*diff;
            opinions(neighbour) = opinions(neighbour) + mu*(-diff);
            % pass update to sorted vector
            altPerson = sorted_opinions(:,2) == person;
            altNeighbour = sorted_opinions(:,2) == neighbour;
            sorted_opinions(altPerson,1) = opinions(person);
            sorted_opinions(altNeighbour,1) = opinions(neighbour);
            sorted_opinions = sortrows( sorted_opinions );
        end
    end

end

end
```

D.2. createRandomSocialGraph.m

Listing 9: createRandomSocialGraph.m

```
function [ M ] = createRandomSocialGraph( n, k )
%createRandomSocialGraphs
%   Create a random sparse adjacency matrix of size (n,n).
%   k specifies the average degree of each node.
% n : Number of people
% k : Number of people known to each person

if nargin < 2, error('Insufficient input arguments.'); end


% Density = peopleKnown / people;
density = k / n;
```

```

% Generate a random symmetric matrix with given density
M = sprandsym(n, density);

% Set all nonzero entries to 1
nz = find(M);
M(nz) = 1;

end
```

D.3. createWattsAndStrogatzModel.m

Listing 10: createWattsAndStrogatzModel.m

```
function [ A ] = createWattsAndStrogatzModel( n, k, beta )
%                        watts and strogatz model
% Create a sparse matrix A representing a undirected graph in the Watts
% and Strogatz model
% see: http://en.wikipedia.org/wiki/Watts_and_Strogatz_model
%
% Input:   n       number of nodes
%          k       mean degree k
%          beta    interpolation parameter between ER graph and regular
%                  ring lattice (0 <= beta <= 1)
%                  beta = 0 : regular ring lattice
%                  beta = 1 : ER graph
% Output:  A       sparse matrix
%
% the parameters must satisfy n >> k >> ln(n) >> 1

% Use default values if no input is given
if nargin < 2
    k = 20;
    beta = 0.5;
end

% error handling
if (mod(k, 2) == 1)
    error('The mean degree k must be divisable by 2.');
end
k2 = k/2;
if (k >= n) % number of neighbours greater than number of nodes
    error('The desired number of neighbours is greater that the number of nodes.');
end

% create sparse nxn matrix A representing regular ring lattice (step 1 wikipedia)
% - row_index holds [1,1,2,2,3,3,... ] always k copies of a number (here 2)
% - col_index holds blocks of of neighbour numbers (modulo n)
%   e.g [8,2, 1,3, 2,4, 3,5, ...]
col_index = zeros(k*n, 1);
for i=1:n
    index = ((i-1)*k+1):(i*k);
    lower = (i-k2-1):(i-2);
    upper = (i):(i+k2-1);
    col_index( index ) = mod([lower, upper], n) + ones(1, k);
end
row_index = reshape(repmat(1:n, k, 1), k*n, 1);
A = sparse(row_index, col_index, ones(n*k, 1), n, n, n*k);

```

```
% For every node (row in the matrix) take every edge (n_i, n_j) with i < j
% and rewire it with probability beta. I.e. we replace (n_i, n_j) with
% (n_i, n_r). (step 2 wikipedia)
for i=1:n
    neighbours = find(A(i,:));
    for j=neighbours
        if (i < j) && (rand < beta)
            r = i;
            % choose until candidate found that avoid loops and link duplication
            while ( (r == i) || (full(A(i,r)) == 1) )
                r = randi(n); % choose random integer number
            end
            A(i,j) = 0; A(j,i) = 0; % reset edge
            A(i,r) = 1; A(r,i) = 1; % new edge
        end
    end
end

% consistency check
if (nnz(A) ~= k*n)
    error(['The number of edges must be k*n/2 and therefore the number'...
    'of nonzero elements n*k']);
end

end
```

D.4. generateContinuousOpinions.m

Listing 11: generateContinuousOpinions.m

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

D.5. generateGraphs.m

Listing 12: generateGraphs.m

```
% Generate graphs for project report

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GRAPHS FOR THE NETWORK-BASED MODEL

%% OUTPUT 1a - Model correctness - Random
%   Equivalent figure to "Figure 2" in paper 1
%   Used to compare discrete model against model from paper 1

clear all;
close all;
clc;

% Set parameters
n = 3200;
k = 4;
```

```
gamma = 10;
average_iterations = 50;
iter = 20000000;
figureFolder = '../documentation/figures/';


fprintf('Random\n');

% Generate values
phi = 0.04;
fprintf('Part 1: phi = %1.3f\n', phi);
dist1 = runModel(n,'random',k,gamma,phi,iter,average_iterations);

phi = 0.458;
fprintf('Part 2: phi = %1.3f\n', phi);
dist2 = runModel(n,'random',k,gamma,phi,iter,average_iterations);

phi = 0.96;
fprintf('Part 3: phi = %1.3f\n', phi);
dist3 = runModel(n,'random',k,gamma,phi,iter,average_iterations);

% Plot graphic
subplot('Position',[.1 .1+1.6/3 .8 .8/3])
loglog(0:n-1,dist1./(n/gamma),'o')
xlim([0 3200])
ylim([10^(-5.5) 0.2])
set(gca,'XTickLabel',{})
set(gca,'YTick',[0.00001 0.0001 0.001 0.01 0.1])
set(gca,'YTickLabel',' |1E-4| |0.01| ')
ylabel('\bf P(s)')
string = '$\textbf{(a)} \qquad \phi = 0.04$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

subplot('Position',[.1 .1+.8/3 .8 .8/3])
loglog(0:n-1,dist2./(n/gamma),'o')
xlim([0 3200])
ylim([10^(-9) 0.2])
set(gca,'XTickLabel',{})
set(gca,'YTick',[0.00000001 0.0000001 0.000001 0.00001 0.0001 0.001 0.01 0.1])
set(gca,'YTickLabel','1E-8| |1E-6| |1E-4| |0.01|')
ylabel('\bf P(s)')
string = '$\textbf{(b)} \qquad \phi = 0.458$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

subplot('Position',[.1 .1 .8 .8/3])
loglog(0:n-1,dist3./(n/gamma),'o')
xlim([0 3200])
ylim([10^(-6) 0.2])
set(gca,'YTick',[0.000001 0.00001 0.0001 0.001 0.01 0.1])
set(gca,'YTickLabel','1E-6| |1E-4| |0.01| ')
set(gca,'XTickLabel',{'1','10','100','1000'})
xlabel('\bf s')
ylabel('\bf P(s)')
string = '$\textbf{(c)} \qquad \phi = 0.96$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

% Save files
filename = [figureFolder,'discrete_model_random'];
```

```
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);


%% OUTPUT 1b - Small world
%  Equivalent to OUTPUT 1a, except that
%  we use the small world system as the initial graph structure

clear all;
close all;
clc;

% Set parameters
n = 3200;
k = 4;
gamma = 10;
average_iterations = 20;
iter = 10000000;
figureFolder = '../documentation/figures/';

fprintf('Small world\n');

% Generate values
phi = 0.04;
fprintf('Part 1: phi = %1.3f\n', phi);
dist1 = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations);

phi = 0.458;
fprintf('Part 2: phi = %1.3f\n', phi);
dist2 = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations);

phi = 0.96;
fprintf('Part 3: phi = %1.3f\n', phi);
dist3 = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations);

% Plot graphic
subplot('Position',[.1 .1+1.6/3 .8 .8/3])
loglog(0:n-1,dist1./(n/gamma),'o')
xlim([0 3200])
ylim([10^(-5.5) 0.2])
set(gca,'YTick',[0.00001 0.0001 0.001 0.01 0.1])
set(gca,'YTickLabel',' |1E-4| |0.01| ')
set(gca,'XTickLabel',{})
ylabel('\bf P(s)')
string = '$\textbf{(a)} \qquad \phi = 0.04$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

subplot('Position',[.1 .1+.8/3 .8 .8/3])
loglog(0:n-1,dist2./(n/gamma),'o')
xlim([0 3200])
ylim([10^(-9) 0.2])
set(gca,'XTickLabel',{})
set(gca,'YTick',[0.00000001 0.0000001 0.000001 0.00001 0.0001 0.001 0.01 0.1])
set(gca,'YTickLabel','1E-8| |1E-6| |1E-4| |0.01|')
ylabel('\bf P(s)')
string = '$\textbf{(b)} \qquad \phi = 0.458$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

subplot('Position',[.1 .1 .8 .8/3])
```

```
loglog(0:n-1,dist3./(n/gamma),'o')
xlim([0 3200])
ylim([10^(-6) 0.2])
set(gca,'YTick',[0.000001 0.00001 0.0001 0.001 0.01 0.1])
set(gca,'YTickLabel','1E-6| |1E-4| |0.01| ')
set(gca,'XTickLabel',{'1','10','100','1000'})
xlabel('\bf s')
ylabel('\bf P(s)')
string = '$\textbf{(c)} \qquad \phi = 0.96$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

% Save files
filename = [figureFolder,'discrete_model_ws'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);
save([figureFolder,'discrete_model_ws_phi_004.mat']);

%% OUTPUT 2a - Convergence / Neccessary iterations for consensus - Random

clear all;
close all;
clc;

% Set parameters
n = 3200;
k = 4;
gamma = 10;
average_iterations = 1;
iter = 0;
max_iterations = 100*10^6;
threshold = 0;
figureFolder = '../documentation/figures/';


% Generate values
phi = 0.04;
d_it = 10^6;
fprintf('Part 1: phi = %1.3f\n', phi);
[~,~,~,diffs1] = runModel(n,'random',k,gamma,phi,iter,average_iterations,...
                     max_iterations,threshold,d_it);
phi = 0.458;
d_it = 10000;
fprintf('Part 2: phi = %1.3f\n', phi);
[~,~,~,diffs2] = runModel(n,'random',k,gamma,phi,iter,average_iterations,...
                        max_iterations,threshold,d_it);
phi = 0.96;
d_it = 10000;
fprintf('Part 3: phi = %1.3f\n', phi);
[~,~,~,diffs3] = runModel(n,'random',k,gamma,phi,iter,average_iterations,...
                        max_iterations,threshold,d_it);

% Plot graphic & save files
plot(diffs1(1,2:end),diffs1(2,2:end),'-')
xlabel('{\bf iterations}','fontsize',12);
ylabel('{\bf normed difference to last step}','fontsize',12);

filename = [figureFolder,'discrete_model_diff_random_004'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);

```

```
plot(diffs2(1,2:end),diffs2(2,2:end),'-')
xlabel('{\bf iterations}','fontsize',12);
ylabel('{\bf normed difference to last step}','fontsize',12);

filename = [figureFolder,'discrete_model_diff_random_0458'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);

plot(diffs3(1,2:end),diffs3(2,2:end),'-')
xlabel('{\bf iterations}','fontsize',12);
ylabel('{\bf normed difference to last step}','fontsize',12);

filename = [figureFolder,'discrete_model_diff_random_096'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);


%% OUTPUT 2b - Convergence / Neccessary iterations for consensus - Random

clear all;
close all;
clc;

% Set parameters
n = 3200;
k = 4;
gamma = 10;
average_iterations = 1;
iter = 0;
max_iterations = 100*10^6;
threshold = 0;
figureFolder = '../documentation/figures/';


% Generate values
phi = 0.04;
d_it = 10^6;
fprintf('Part 1: phi = %1.3f\n', phi);
[~,~,~,diffs1] = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations,...
                    max_iterations,threshold,d_it);
phi = 0.458;
d_it = 10000;
fprintf('Part 2: phi = %1.3f\n', phi);
[~,~,~,diffs2] = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations,...
                        max_iterations,threshold,d_it);
phi = 0.96;
d_it = 10000;
fprintf('Part 3: phi = %1.3f\n', phi);
[~,~,~,diffs3] = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations,...
                        max_iterations,threshold,d_it);

% Plot graphic & save files
plot(diffs1(1,2:end),diffs1(2,2:end),'-')
xlabel('{\bf iterations}','fontsize',12);
ylabel('{\bf normed difference to last step}','fontsize',12);

filename = [figureFolder,'discrete_model_diff_ws_004'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);

plot(diffs2(1,2:end),diffs2(2,2:end),'-')
```

```
xlabel('{\bf iterations}','fontsize',12);
ylabel('{\bf normed difference to last step}','fontsize',12);

filename = [figureFolder,'discrete_model_diff_ws_0458'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);

plot(diffs3(1,2:end),diffs3(2,2:end),'-')
xlabel('{\bf iterations}','fontsize',12);
ylabel('{\bf normed difference to last step}','fontsize',12);

filename = [figureFolder,'discrete_model_diff_ws_096'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);


%% OUTPUT 3a - Opinion developement graphs - Random
clear all;
close all;
clc;

% Set parameters
n = 3200;
k = 4;
gamma = 10;
iter = 0;
average_iterations = 1;
max_iterations = 20000000;
threshold = 0;
figureFolder = '../documentation/figures/';

fprintf('Random\n');


% Generate values
phi = 0.04;
d_it = 10^5;
fprintf('Part 1: phi = %1.3f\n', phi);
[~,ops1,~] = runModel(n,'random',k,gamma,phi,iter,average_iterations,...
                  max_iterations,threshold,d_it);
phi = 0.458;
d_it = 1000;
fprintf('Part 2: phi = %1.3f\n', phi);
[~,ops2,~] = runModel(n,'random',k,gamma,phi,iter,average_iterations,...
                 max_iterations,threshold,d_it);
phi = 0.96;
d_it = 1000;
fprintf('Part 3: phi = %1.3f\n', phi);
[~,ops3,~] = runModel(n,'random',k,gamma,phi,iter,average_iterations,...
                  max_iterations,threshold,d_it);

% Plot graphic & save files
pcolor(ops1);
shading flat;
xlabel('{\bf x10^5 iterations}','fontsize',12);
ylabel('{\bf opinions}','fontsize',12);
set(gca,'YTickLabel',{})

filename = [figureFolder,'discrete_opinion_developement_random_004'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);
```

```
pcolor(ops2);
shading flat;
xlabel('{\bf x10^3 iterations}','fontsize',12);
ylabel('{\bf opinions}','fontsize',12);
set(gca,'YTickLabel',{})

filename = [figureFolder,'discrete_opinion_developement_random_0458'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);



pcolor(ops3);
shading flat;
xlabel('{\bf x10^3 iterations}','fontsize',12);
ylabel('{\bf opinions}','fontsize',12);
set(gca,'YTickLabel',{})

filename = [figureFolder,'discrete_opinion_developement_random_096'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);


surf(ops1);
shading interp;
axis tight
xlabel('{\bf x10^5 iterations}','fontsize',12);
ylabel('{\bf opinions}','fontsize',12);
zlabel('{\bf group sizes}','fontsize',12);
set(gca,'YTickLabel',{})

filename = [figureFolder,'discrete_opinion_developement_random_3d_004'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);


surf(ops2);
shading interp;
axis tight
xlabel('{\bf x10^3 iterations}','fontsize',12);
ylabel('{\bf opinions}','fontsize',12);
zlabel('{\bf group sizes}','fontsize',12);
set(gca,'YTickLabel',{})

filename = [figureFolder,'discrete_opinion_developement_random_3d_0458'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);


surf(ops3);
shading interp;
axis tight
xlabel('{\bf x10^3 iterations}','fontsize',12);
ylabel('{\bf opinions}','fontsize',12);
zlabel('{\bf group sizes}','fontsize',12);
set(gca,'YTickLabel',{})

filename = [figureFolder,'discrete_opinion_developement_random_3d_096'];
```

```
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);


%% OUTPUT 3b - Opinion developement graphs - Small world

clear all;
close all;
clc;

% Set parameters
n = 3200;
k = 4;
gamma = 10;
iter = 0;
average_iterations = 1;
max_iterations = 20000000;
threshold = 0;
figureFolder = '../documentation/figures/';

fprintf('Small world\n');

% Generate values
phi = 0.04;
d_it = 10^5;
fprintf('Part 1: phi = %1.3f\n', phi);
[~,ops1,~] = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations,...
                    max_iterations,threshold,d_it);
phi = 0.458;
d_it = 1000;
fprintf('Part 2: phi = %1.3f\n', phi);
[~,ops2,~] = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations,...
                   max_iterations,threshold,d_it);
phi = 0.96;
d_it = 1000;
fprintf('Part 3: phi = %1.3f\n', phi);
[~,ops3,~] = runModel(n,'watts_strogatz',k,gamma,phi,iter,average_iterations,...
                    max_iterations,threshold,d_it);

% Plot graphic & save files
pcolor(ops1);
shading flat;
xlabel('{\bf x10^5 iterations}','fontsize',12);
ylabel('{\bf opinions}','fontsize',12);
set(gca,'YTickLabel',{})
title('\bf Opinion distribution over time. $\phi = 0.04$'....
                              ,'interpreter','latex','fontsize',12);

filename = [figureFolder,'discrete_opinion_developement_ws_004'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);



pcolor(ops2);
shading flat;
xlabel('{\bf x10^3 iterations}','fontsize',12);
ylabel('{\bf opinions}','fontsize',12);
set(gca,'YTickLabel',{})
title('\bf Opinion distribution over time. $\phi = 0.458$'....
                              ,'interpreter','latex','fontsize',12);
```

```
filename = [figureFolder,'discrete_opinion_developement_ws_0458'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);



pcolor(ops3);
shading flat;
xlabel('{\bf x10^3 iterations}','fontsize',12);
ylabel('{\bf opinions}','fontsize',12);
set(gca,'YTickLabel',{})
title('\bf Opinion distribution over time. $\phi = 0.96$'....
                              ,'interpreter','latex','fontsize',12);

filename = [figureFolder,'discrete_opinion_developement_ws_096'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);


surf(ops1);
shading interp;
axis tight
xlabel('{\bf x10^5 iterations}','fontsize',12);
ylabel('{\bf opinions}','fontsize',12);
zlabel('{\bf group sizes}','fontsize',12);
set(gca,'YTickLabel',{})
title('\bf Opinion distribution over time. $\phi = 0.04$'....
                              ,'interpreter','latex','fontsize',12);

filename = [figureFolder,'discrete_opinion_developement_ws_3d_004'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);


surf(ops2);
shading interp;
axis tight
xlabel('{\bf x10^3 iterations}','fontsize',12);
ylabel('{\bf opinions}','fontsize',12);
zlabel('{\bf group sizes}','fontsize',12);
set(gca,'YTickLabel',{})
title('\bf Opinion distribution over time. $\phi = 0.458$'....
                              ,'interpreter','latex','fontsize',12);

filename = [figureFolder,'discrete_opinion_developement_ws_3d_0458'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);


surf(ops3);
shading interp;
axis tight
xlabel('{\bf x10^3 iterations}','fontsize',12);
ylabel('{\bf opinions}','fontsize',12);
zlabel('{\bf group sizes}','fontsize',12);
set(gca,'YTickLabel',{})
title('\bf Opinion distribution over time. $\phi = 0.96$'....
                              ,'interpreter','latex','fontsize',12);

filename = [figureFolder,'discrete_opinion_developement_ws_3d_096'];
```

```
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);


%% OUTPUT 4 - examine dependency on beta (parameter in Watts and Strogatz)
%   Generated graphs examine the dependency on beta of the group size
%   distribution. For that several graphs equivalent to the one of Figure 2
%   in paper [1] are generated with different betas.

% reset
clear all;
close all;
clc;
if (matlabpool('size') ~= 0) % closes open workers
    matlabpool close force;
end

% Set values
n = 3200;
k = 4;
gamma = 10;
average_iterations = 10;
iter = 20000000;
max_iterations = 500000;
threshold = 0.05;
d_it = 10000;

% The simulation is run for the following values (betas).
betas = 0:0.5:1;

% folder to save the figures
figureFolder = '../documentation/figures/';

% flag to turn the parallel toolbox on or off
parallel = false;

fprintf('Small world\n\n');

% Setup matlabpool workers to run job in parallel if demanded.
% (default configuration is used for matlabpool)
if parallel
    matlabpool open;
end

for i = 1:size(betas,2)
    fprintf('beta: %0.2f\n', betas(i));

    % running simulation for several phi's
    phi = 0.04;
    fprintf('Part 1: phi = %1.3f\n', phi)
    tic; dist1(:,i) = runModel(n,'watts_strogatz',k,gamma,phi,iter,...
        average_iterations,max_iterations,threshold,d_it,betas(i)); toc;

    phi = 0.458;
    fprintf('Part 2: phi = %1.3f\n', phi)
    tic; dist2(:,i) = runModel(n,'watts_strogatz',k,gamma,phi,iter,...
        average_iterations,max_iterations,threshold,d_it,betas(i)); toc;

    phi = 0.96;
    fprintf('Part 3: phi = %1.3f\n', phi)
    tic; dist3(:,i) = runModel(n,'watts_strogatz',k,gamma,phi,iter,...
```

```
        average_iterations,max_iterations,threshold,d_it,betas(i)); toc;

    % plotting
    subplot(3,1,1)

    subplot('Position',[.1 .1+1.6/3 .8 .8/3])
    loglog(1:n,dist1(:,i)./(n/gamma),'o')
    xlim([0 3200])
    set(gca,'XTickLabel',{})
    set(gca,'YTickLabel',' |0.01|0.1| ')
    ylabel('P(s)')
    string = '$\textbf{(a)} \qquad \phi = 0.04$';
    text('string',string,'position',[0.7 0.8],'interpreter','latex',...
        'fontsize',14,'units','norm')

    subplot('Position',[.1 .1+.8/3 .8 .8/3])
    loglog(1:n,dist2(:,i)./(n/gamma),'o')
    xlim([0 3200])
    set(gca,'XTickLabel',{})
    set(gca,'YTickLabel',' |0.01|0.1| ')
    ylabel('P(s)')
    string = '$\textbf{(b)} \qquad \phi = 0.458$';
    text('string',string,'position',[0.7 0.8],'interpreter','latex',...
        'fontsize',14,'units','norm')

    subplot('Position',[.1 .1 .8 .8/3])
    loglog(1:n,dist3(:,i)./(n/gamma),'o')
    xlim([0 3200])
    set(gca,'XTickLabel',{'1','10','100','1000'})
    set(gca,'YTickLabel',' |0.01|0.1| ')
    xlabel('s')
    ylabel('P(s)')
    string = '$\textbf{(c)} \qquad \phi = 0.96$';
    text('string',string,'position',[0.7 0.8],'interpreter','latex',...
        'fontsize',14,'units','norm')

    % saving the figures in different formats
    filename = [figureFolder,'discrete_model_ws_k4_beta_', sprintf('%.2f',betas(i))];
    print('-dpng', [filename,'.png']);
    print('-depsc2', [filename,'.eps']);
    saveas(gcf, [filename, '.fig']);

    fprintf('\n');

end

% saving all the variables used for plotting in a file
save([figureFolder,'discrete_model_ws_k4_beta_05.mat']);

% close matlabpool workers if job was run in parallel
if parallel
    matlabpool close;
end

%% OUTPUT 5 - examine dependency on beta and speed of convergence
%   Generated graphs examine the dependency on beta on the speed of
%   convergence. For that the number of iterations require to fall below
%   the specified threshold are calculated for several betas and all the
%   three values for phi.

% reset
```

```matlab
clear all;
close all;
clc;
if (matlabpool('size') ~= 0) % closes open workers
    matlabpool close force;
end

% Set values
n = 3200;
k = 4;
gamma = 10;
average_iterations = 10;
iter = 0;
max_iterations = 20000000;
threshold = 0.02;
d_it = 500;

% The simulation is run for the following values (betas).
betas = 0.0:0.5:1;

% folder to save the figures
figureFolder = '../documentation/figures/';


% flag to turn the parallel toolbox on or off
parallel = true;

fprintf('Small world\n\n');

% Setup matlabpool workers to run job in parallel if demanded.
% (default configuration is used for matlabpool)
if parallel
    matlabpool open;
end

iterations_1 = zeros(size(betas,2),1);
iterations_2 = zeros(size(betas,2),1);
iterations_3 = zeros(size(betas,2),1);


parfor i = 1:size(betas,2)

    fprintf('beta: %0.2f\n', betas(i));

    phi = 0.04;
    fprintf('Part 1: phi = %1.3f\n', phi)
    tic; [~,~,iterations_1(i),~] = runModel(n,'watts_strogatz',k,gamma,phi,...
        iter,average_iterations,max_iterations,threshold,d_it,betas(i)); toc;
    phi = 0.458;
    fprintf('Part 2: phi = %1.3f\n', phi)
    tic; [~,~,iterations_2(i),~] = runModel(n,'watts_strogatz',k,gamma,phi,...
        iter,average_iterations,max_iterations,threshold,d_it,betas(i)); toc;
    phi = 0.96;
    fprintf('Part 3: phi = %1.3f\n', phi)
    tic; [~,~,iterations_3(i),~] = runModel(n,'watts_strogatz',k,gamma,phi,...
        iter,average_iterations,max_iterations,threshold,d_it,betas(i)); toc;

    fprintf('\n');
end

% plotting
```

```
plot(betas, [iterations_1, iterations_2, iterations_3]);
xlabel('{\bf \beta}','fontsize',12);
ylabel('{\bf number of iterations}','fontsize',12);
legend('\phi = 0.04', '\phi = 0.458', '\phi = 0.96');

% saving the figures in different formats
filename = [figureFolder,'discrete_model_ws_k4_number_iterations'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);
saveas(gcf, [filename, '.fig']);

% saving all the variables used for plotting in a file
save([figureFolder,'discrete_model_ws_k4_number_iterations_all.mat']);

% close matlabpool workers if job was run in parallel
if parallel
    matlabpool close
end

%% OUTPUT 6 - examine dependency on beta and speed of convergence
%   Generated graphs examine the dependency on beta on the speed of
%   convergence. For that the number of iterations require to fall below
%   the specified threshold are calculated for several betas and phi = 0.04

% reset
clear all;
close all;
clc;
if (matlabpool('size') ~= 0) % closes open workers
    matlabpool close force;
end

% Set values
n = 3200;
k = 4;
gamma = 10;
average_iterations = 1;
iter = 0;
max_iterations = 20000000;
threshold = 0.01;
d_it = 500;

% The simulation is run for the following values (betas).
betas = linspace(0,1,16);

% folder to save the figures
figureFolder = '../documentation/figures/';


% flag to turn the parallel toolbox on or off
parallel = true;

fprintf('Small world\n\n');

% Setup matlabpool workers to run job in parallel if demanded.
% (default configuration is used for matlabpool)
if parallel
    matlabpool open;
end

iterations = zeros(size(betas,2),1);
```

```
parfor i = 1:size(betas,2)

    fprintf('beta: %0.2f\n', betas(i));

    phi = 0.04;
    % fprintf('phi = %1.3f\n', phi)
    tic; [~,~,iterations(i),~] = runModel(n,'watts_strogatz',k,gamma,phi,...
        iter,average_iterations,max_iterations,threshold,d_it,betas(i)); toc;

    fprintf('\n');
end

% plotting
plot(betas,iterations);
xlabel('{\bf \beta}','fontsize',12);
ylabel('{\bf number of iterations}','fontsize',12);
legend('\phi = 0.04');

% saving the figures in different formats
filename = [figureFolder,'discrete_model_ws_k4_number_iterations'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);
saveas(gcf, [filename, '.fig']);

% saving all the variables used for plotting in a file
save([figureFolder,'discrete_model_ws_k4_number_iterations_all.mat']);

% close matlabpool workers if job was run in parallel
if parallel
    matlabpool close
end

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GRAPHS FOR THE CONTINUOUS MODEL

%% OUTPUT 1
%   Equivalent to Fig 1 in paper 2

% reset
clear all; close all; clc;
figure_handle = figure;

% setup the combined model to simulate the continuous model
graph = 'complete';
k = 0;
phi = 0;
std = 0;

% set parameters according to paper 2
n=3200;
% Paper: n = 500000;
% This requires to much space as the combined model allocates space
% for a complete graph which is n^2 space
u = 0.35;
mu = 0.001;
clusters = 1000; % u_0 = 0.001, clusters = 1/u_0
iterations = 400*n;
cskip = 400;

iterations = 80000; mu = 0.3; clusters = 100; cskip = 300;
```

```
% simulate
[histogram] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations);

% graph
pcolor( histogram(1:end-1,:) );
shading flat;
title( 'Opinion distribution over time' );
ylabel( 'Opinion' );
xlabel( 't/N' );

% save results
figureFolder = '../documentation/figures/';
filename = [figureFolder,'continuous_model'];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);
saveas(figure_handle, [filename, '.fig']);

%% OUTPUT 2
%   Examine the 1/2u behaviour

% reset
clear all; close all; clc;
parallel = false;

% setup the combined model to simulate the continuous model
graph = 'complete';
k = 0;
phi = 0;
std = 0;

% set parameters for the continuous model
iterations = 80000;
mu = 0.3;
clusters = 100;
cskip = 250;
n = 3200;

% define the expected opinion clusters depending on u;
expected_clusters = [
    1/(2*0.3); % minimum u to achieve consensus according to paper 2
    2;
    5;
    8;
    15 ];
expected_clusters(:,2) = 1./( 2 * expected_clusters(:,1) );

% initialize parallel computation
if parallel
    matlabpool open
end

parfor i = 1:size(expected_clusters,1)
    % simulate
    e = expected_clusters(i,1);
    u = expected_clusters(i,2);
    [histogram] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations);
    % graph
    figure_handle = figure;
    pcolor( histogram(1:end-1,:) );
    shading flat;
```

```
    title( sprintf('Opinion distribution over time. u = %2.2f',u) );
    ylabel( 'Opinion' );
    xlabel( 't/N' );

    % save results
    figureFolder = '../documentation/figures/';
    filename = [figureFolder, sprintf('continuous_model_u_%2.2f',e) ];
    print('-dpng', [filename,'.png']);
    print('-depsc2', [filename,'.eps']);
    saveas(figure_handle, [filename, '.fig']);
end

% cleanup
if parallel
    matlabpool close
end

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GRAPHS FOR THE COMBINED MODEL

%% OUTPUT 1
%   Generate a figure equivalent to the discrete model.

% reset
clear all; close all; clc;
parallel = false;
figure_handle = figure;

% configure the model
iterations = 1000000;
average_iterations = 10;
n = 3200;
k = 4;
clusters = 320;
cskip = 100;
graph = 'random';
Phi = [ 0.04, 0.458, 0.96];
std = 0.1;
u = 0.3;
mu = 0.3;

% Generate values
dist = zeros(n,average_iterations);

phi = Phi(1);
fprintf('Part 1: phi = %1.3f\n', phi);
for i=1:average_iterations
    fprintf('Average %i/%i\n',i,average_iterations);
    [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations);
end
dist1 = sum(dist,2)./average_iterations;

phi = Phi(2);
fprintf('Part 2: phi = %1.3f\n', phi);
for i=1:average_iterations
    fprintf('Average %i/%i\n',i,average_iterations);
    [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations);
end
dist2 = sum(dist,2)./average_iterations;

phi = Phi(3);
```

```
fprintf('Part 3: phi = %1.3f\n', phi);
for i=1:average_iterations
    fprintf('Average %i/%i\n',i,average_iterations);
    [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations);
end
dist3 = sum(dist,2)./average_iterations;

% Plot graphic
subplot('Position',[.1 .1+1.6/3 .8 .8/3])
loglog(1:n,dist1./(clusters),'o')
xlim([0 3200])
ylim([10^(-5.5) 0.2])
set(gca,'XTickLabel',{})
ylabel('P(s)')
string = '$\textbf{(a)} \qquad \phi = 0.04$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

subplot('Position',[.1 .1+.8/3 .8 .8/3])
loglog(1:n,dist2./(clusters),'o')
xlim([0 3200])
ylim([10^(-9) 0.2])
set(gca,'XTickLabel',{})
%set(gca,'YTickLabel',' |0.01|0.1| ')
ylabel('P(s)')
string = '$\textbf{(b)} \qquad \phi = 0.458$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

subplot('Position',[.1 .1 .8 .8/3])
loglog(1:n,dist3./(clusters),'o')
xlim([0 3200])
ylim([10^(-6) 0.2])
set(gca,'XTickLabel',{'1','10','100','1000'})
xlabel('s')
ylabel('P(s)')
string = '$\textbf{(c)} \qquad \phi = 0.96$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

% save results
figureFolder = '../documentation/figures/';
filename = [figureFolder, 'combined_model_random' ];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);
saveas(figure_handle, [filename, '.fig']);

%% OUTPUT 2
%  Same as output 1, but for watts and strogatz graph beta=0.25

% reset
clear all; close all; clc;
parallel = false;
figure_handle = figure;

% configure the model
iterations = 1000000;
average_iterations = 10;
n = 3200;
k = 4;
clusters = 320;
```

```
cskip = 100;
graph = 'watts_strogatz';
Phi = [ 0.04, 0.458, 0.96];
std = 0.1;
u = 0.3;
mu = 0.3;

beta = 0.25;

% Generate values
dist = zeros(n,average_iterations);

phi = Phi(1);
fprintf('Part 1: phi = %1.3f\n', phi);
for i=1:average_iterations
    fprintf('Average %i/%i\n',i,average_iterations);
    [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
end
dist1 = sum(dist,2)./average_iterations;

phi = Phi(2);
fprintf('Part 2: phi = %1.3f\n', phi);
for i=1:average_iterations
    fprintf('Average %i/%i\n',i,average_iterations);
    [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
end
dist2 = sum(dist,2)./average_iterations;

phi = Phi(3);
fprintf('Part 3: phi = %1.3f\n', phi);
for i=1:average_iterations
    fprintf('Average %i/%i\n',i,average_iterations);
    [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
end
dist3 = sum(dist,2)./average_iterations;

% Plot graphic
subplot('Position',[.1 .1+1.6/3 .8 .8/3])
loglog(1:n,dist1./(clusters),'o')
xlim([0 3200])
ylim([10^(-5.5) 0.2])
set(gca,'XTickLabel',{})
ylabel('P(s)')
string = '$\textbf{(a)} \qquad \phi = 0.04$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

subplot('Position',[.1 .1+.8/3 .8 .8/3])
loglog(1:n,dist2./(clusters),'o')
xlim([0 3200])
ylim([10^(-9) 0.2])
set(gca,'XTickLabel',{})
%set(gca,'YTickLabel',' |0.01|0.1| ')
ylabel('P(s)')
string = '$\textbf{(b)} \qquad \phi = 0.458$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

subplot('Position',[.1 .1 .8 .8/3])
loglog(1:n,dist3./(clusters),'o')
xlim([0 3200])
```

```
ylim([10^(-6) 0.2])
set(gca,'XTickLabel',{'1','10','100','1000'})
xlabel('s')
ylabel('P(s)')
string = '$\textbf{(c)} \qquad \phi = 0.96$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

% save results
figureFolder = '../documentation/figures/';
filename = [figureFolder, 'combined_model_watts_strogatz_beta_025' ];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);
saveas(figure_handle, [filename, '.fig']);

%% OUTPUT 3
%   Same as output 1, but for watts and strogatz graph beta=0.5

% reset
clear all; close all; clc;
parallel = false;
figure_handle = figure;

% configure the model
iterations = 1000000;
average_iterations = 10;
n = 3200;
k = 4;
clusters = 320;
cskip = 100;
graph = 'watts_strogatz';
Phi = [ 0.04, 0.458, 0.96];
std = 0.1;
u = 0.3;
mu = 0.3;

beta = 0.5;

% Generate values
dist = zeros(n,average_iterations);

phi = Phi(1);
fprintf('Part 1: phi = %1.3f\n', phi);
for i=1:average_iterations
    fprintf('Average %i/%i\n',i,average_iterations);
    [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
end
dist1 = sum(dist,2)./average_iterations;

phi = Phi(2);
fprintf('Part 2: phi = %1.3f\n', phi);
for i=1:average_iterations
    fprintf('Average %i/%i\n',i,average_iterations);
    [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
end
dist2 = sum(dist,2)./average_iterations;

phi = Phi(3);
fprintf('Part 3: phi = %1.3f\n', phi);
for i=1:average_iterations
    fprintf('Average %i/%i\n',i,average_iterations);
```

```
    [tmp dist(:,i)] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
end
dist3 = sum(dist,2)./average_iterations;

% Plot graphic
subplot('Position',[.1 .1+1.6/3 .8 .8/3])
loglog(1:n,dist1./(clusters),'o')
xlim([0 3200])
ylim([10^(-5.5) 0.2])
set(gca,'XTickLabel',{})
ylabel('P(s)')
string = '$\textbf{(a)} \qquad \phi = 0.04$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

subplot('Position',[.1 .1+.8/3 .8 .8/3])
loglog(1:n,dist2./(clusters),'o')
xlim([0 3200])
ylim([10^(-9) 0.2])
set(gca,'XTickLabel',{})
%set(gca,'YTickLabel',' |0.01|0.1| ')
ylabel('P(s)')
string = '$\textbf{(b)} \qquad \phi = 0.458$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

subplot('Position',[.1 .1 .8 .8/3])
loglog(1:n,dist3./(clusters),'o')
xlim([0 3200])
ylim([10^(-6) 0.2])
set(gca,'XTickLabel',{'1','10','100','1000'})
xlabel('s')
ylabel('P(s)')
string = '$\textbf{(c)} \qquad \phi = 0.96$';
text('string',string,'position',[0.7 0.8],'interpreter','latex',...
    'fontsize',14,'units','norm')

% save results
figureFolder = '../documentation/figures/';
filename = [figureFolder, 'combined_model_watts_strogatz_beta_050' ];
print('-dpng', [filename,'.png']);
print('-depsc2', [filename,'.eps']);
saveas(figure_handle, [filename, '.fig']);


%% OUTPUT 4
%   Examine the 1/2u behaviour in the combined model for a random graph

% reset
clear all; close all; clc;

% configure the model
iterations = 1000000;
average_iterations = 10;
n = 3200;
k = 4;
clusters = 320;
cskip = 100;
graph = 'random';
Phi = [ 0.04, 0.458, 0.96];
std = 0.1;
```

```
u = 0.3;
mu = 0.3;

% define the expected opinion clusters depending on u;
expected_clusters = [
    1/(2*0.3); % minimum u to achieve consensus according to paper 2
    2;
    5;
    8;
    15 ];
expected_clusters(:,2) = 1./( 2 * expected_clusters(:,1) );

for avg=1:average_iterations

for phi=Phi

for i = 1:size(expected_clusters,1)
    % simulate
    e = expected_clusters(i,1);
    u = expected_clusters(i,2);
    fprintf('Iteration %i/%i, phi=%1.3f, e=%2.2f\n',avg,average_iterations,phi,e);
    [histogram] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations);
    % graph
    figure_handle = figure;
    pcolor( histogram(1:end-1,:) );
    shading flat;
    title( sprintf('Opinion distribution over time. u = %2.2f',u) );
    ylabel( 'Opinion' );
    xlabel( 't/N' );

    % save results
    figureFolder = '../documentation/figures/';
    filename = [figureFolder, sprintf('combined_model_random_u_%2.2f_phi_%1.3f_i_%i',e,phi,avg)];
    print('-dpng', [filename,'.png']);
    print('-depsc2', [filename,'.eps']);
    saveas(figure_handle, [filename, '.fig']);
    close(figure_handle);
end

end

end

%% OUTPUT 5
%   Examine the 1/2u behaviour in the combined model for
%   watts strogatz, beta = 0.25

% reset
clear all; close all; clc;

% configure the model
iterations = 1000000;
average_iterations = 10;
n = 3200;
k = 4;
clusters = 320;
cskip = 100;
graph = 'watts_strogatz';
Phi = [ 0.04, 0.458, 0.96];
std = 0.1;
u = 0.3;
```

```
mu = 0.3;

beta = 0.25;

% define the expected opinion clusters depending on u;
expected_clusters = [
    1/(2*0.3); % minimum u to achieve consensus according to paper 2
    2;
    5;
    8;
    15 ];
expected_clusters(:,2) = 1./( 2 * expected_clusters(:,1) );

for avg=1:average_iterations

for phi=Phi

for i = 1:size(expected_clusters,1)
    % simulate
    e = expected_clusters(i,1);
    u = expected_clusters(i,2);
    fprintf('Iteration %i/%i, phi=%1.3f, e=%2.2f\n',avg,average_iterations,phi,e);
    [histogram] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
    % graph
    figure_handle = figure;
    pcolor( histogram(1:end-1,:) );
    shading flat;
    title( sprintf('Opinion distribution over time. u = %2.2f',u) );
    ylabel( 'Opinion' );
    xlabel( 't/N' );

    % save results
    figureFolder = '../documentation/figures/';
    filename = [figureFolder, ...
        sprintf('combined_model_watts_strogatz_025_u_%2.2f_phi_%1.3f_i_%i',e,phi,avg) ];
    print('-dpng', [filename,'.png']);
    print('-depsc2', [filename,'.eps']);
    saveas(figure_handle, [filename, '.fig']);
    close(figure_handle);
end

end

end

%% OUTPUT 6
%   Examine the 1/2u behaviour in the combined model for
%   watts strogatz, beta = 0.5

% reset
clear all; close all; clc;

% configure the model
iterations = 1000000;
average_iterations = 10;
n = 3200;
k = 4;
clusters = 320;
cskip = 100;
graph = 'watts_strogatz';
Phi = [ 0.04, 0.458, 0.96];
```

```
std = 0.1;
u = 0.3;
mu = 0.3;

beta = 0.5;

% define the expected opinion clusters depending on u;
expected_clusters = [
    1/(2*0.3); % minimum u to achieve consensus according to paper 2
    2;
    5;
    8;
    15 ];
expected_clusters(:,2) = 1./( 2 * expected_clusters(:,1) );

for avg=1:average_iterations

for phi=Phi

for i = 1:size(expected_clusters,1)
    % simulate
    e = expected_clusters(i,1);
    u = expected_clusters(i,2);
    fprintf('Iteration %i/%i, phi=%1.3f, e=%2.2f\n',avg,average_iterations,phi,e);
    [histogram] = runContinuousModel(n,graph,k,phi,u,mu,std,clusters,cskip,iterations,beta);
    % graph
    figure_handle = figure;
    pcolor( histogram(1:end-1,:) );
    shading flat;
    title( sprintf('Opinion distribution over time. u = %2.2f',u) );
    ylabel( 'Opinion' );
    xlabel( 't/N' );

    % save results
    figureFolder = '../documentation/figures/';
    filename = [figureFolder, ...
        sprintf('combined_model_watts_strogatz_050_u_%2.2f_phi_%1.3f_i_%i',e,phi,avg) ];
    print('-dpng', [filename,'.png']);
    print('-depsc2', [filename,'.eps']);
    saveas(figure_handle, [filename, '.fig']);
    close(figure_handle);
end

end

end
```

D.6. generateOpinions.m

Listing 13: generateOpinions.m

```
function opinions = generateOpinions( n, factor )
%generateOpinions
%   Generate a random opinion vector of length n.
%   Factor can be used to specify the proportion of opinions in relation to
%   n, where number_of_opinions = n / factor.

if nargin < 2, factor = 10; end

```

```
% Calculate the requested amount of opinions
count = n/factor;

% Generate the pseudorandom opinions
opinions = zeros(n,1);

for i = 1:n
    opinions(i) = randi(count);
end

end
```

D.7. model.m

Listing 14: model.m

```
function [people, opinions] = model(people,opinions,phi,iter)
%model
%   Model our system with given input parameters.
%
% At each step we select one person randomly.
% Then we use METHOD 1 with probability phi or METHOD 2 otherwise,
% to change the opinion or neighbourhood of the selected person.
%
% METHOD 1
%   Select at random one of the edges attached to the selected person
%   and move the other end of that edge to a vertex chosen
%   randomly from the set off all vertices having the same
%   opinion as the selected person.
%
% METHOD 2
%   Pick a random neighbour of the selected person and set the opinion
%   of the person to that of the selected neighbour.
%
% INPUT
%   people     Adjacency matrix representing the system
%
%   opinions   Vector of the agents opinions
%
%   phi        Probability factor for model
%
%   iter       Number of iterations to perform
%
% OUTPUT
%   people     Adjecency matrix after the model has been run
%
%   opinions   Opinions after the model has been run


if nargin < 4, error('Insufficient input arguments.'); end


% Number of people in the system
n = length(opinions);


% Main loop
for t=1:iter
    % Select a person randomly
```

```
    person = randi(n);


    % Check if the person is still connected to someone
    % Do nothing otherwise
    deg = nnz(people(person,:));
    if deg == 0, continue; end


    % Find all neighbours connected to person
    neighbours = find(people(person,:));

    % And select a neighbour at random
    j = randi(length(neighbours));
    neighbour = neighbours(j);

    if neighbour == person
        continue;
    end

    % Chose method 1 or 2 with probability phi
    if rand() <= phi
       % METHOD 1

       % Find all people with the same opinion as person
       opinion_group = find(opinions == opinions(person));

       % Select one of the people with the same opinion at random
       j = randi(length(opinion_group));
       new_neighbour = opinion_group(j);

       % Connect person and new_neighbour
       % Disconnect person and old_neighbour
       if person ~= new_neighbour
           people(person,new_neighbour) = 1;
           people(new_neighbour,person) = 1;
           people(person,neighbour) = 0;
           people(neighbour,person) = 0;
       end
    else
       % METHOD 2

       % Adopt opinion of neighbour
       opinions(person) = opinions(neighbour);
    end
end

end
```

D.8. runContinousModel.m

Listing 15: runContinuousModel.m

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

```
%               'random', 'watts_strogatz', 'complete'
%   k           average number of connections per person
%   phi         probability factor for model
%   u           opinion threshold
%   mu          convergence parameter
%   std         standard derivation to find new friends by opinion
%   clusters    number of opinion clusters to observe
%   cskip       skip factor for the returned histogram with respect to
%               the number of iterations
%   iterations number of iterations
%   beta        watts strogatz parameter
% OUTPUT
%   histogram  histogram according to the continuousModel function
%   groups_distribution
%               vector where the i-th component is the amount of
%               of opinion groups of that size in the simulated system

% global DEBUG;

% input cosmetics
if nargin < 1, warning('Using defaults!'), n = 3200; end
if nargin < 2, graph = 'watts_strogatz'; end
if nargin < 3, k = 2; end
if nargin < 4, phi = 0.4; end
if nargin < 5, u = 0.04; end
if nargin < 6, mu = 0.3; end
if nargin < 7, std = 0.1; end
if nargin < 8, clusters = 100; end
if nargin < 9, cskip = 300; end
if nargin < 10, iterations = 1000000; end
if nargin < 11, beta = 0.25; end

% graph and opinion initialization
if strcmp(graph,'random')
    people = createRandomSocialGraph(n,k);
elseif strcmp(graph,'watts_strogatz')
    people = createWattsAndStrogatzModel(n,k,beta);
elseif strcmp(graph,'complete')
    people = ones(n,n);
else
    error('Unknown Social Graph');
end
opinions = generateContinuousOpinions(n);

% run the simulation with the specified parameters
[people, opinions, histogram] = ...
    continuousModel(people,opinions,phi,u,mu,std,iterations,clusters,cskip);

% Overview of group sizes (how many groups of each size exist)
% partition the continuous opinions into #clusters opinion groups,
% get the distribution
opinion_distribution = hist(histogram(1:end-1,end),clusters);
% evaluate how many groups have the same size
groups_distribution = hist(opinion_distribution,1:n)';

% generate graphs only if the caller doesn't use the data
if nargout > 0
    return
end

% plot the change of opinions over time
```

```
subplot(2,2,1);
pcolor(histogram(1:end-1,:));
shading flat;
title('Opinion distribution over time');

% plot the normed change in #cskip iterations over time
subplot(2,2,3);
plot(histogram(end,:));
ylim([0 30]);
title('Normed change during the iterations');

% plot the opinion distribution at the end
subplot(2,2,2);
hist(opinions,clusters);
title('Opinion distribution at the end');

% group size distribution
subplot(2,2,4);
loglog(groups_distribution,'o');
title('Group size distribution');

% 3D plot of the histogram. This is plotted in a new window
set(0,'CurrentFigure',figure)
surf(histogram(1:end-1,:));
shading interp;
colormap('Hot');

end
```

D.9. runModel.m

Listing 16: runModel.m

```
function [mean_distribution opinion_dist mean_number_iterations step_differences]...
                                = runModel(n,graph,k,gamma,phi,...
                                iterations, average_iterations,...
                                max_iterations, threshold, d_it, beta)
%runModel
%   Simulate a number of steps of the model with the supplied parameters
%   to compute the opinion group size distribution, i.e. how many opinion
%   groups of sizes 1,...,n exist at the end of the simulation.
%
%   We optionally average over several iterations.
%   If iterations=0 is supplied, the function will run the simulation in
%   steps of 10000 iterations and terminate as soon as the normed difference
%   between the last two distributions reaches a certain threshold.
%   In this case the opinion distributions are also aggregated in betweeen
%   runs and returned in opinion_dist.
%
% INPUT
%   n                 Number of agents in the system
%
%   graph             Use random network ('random')
%                     or small world ('watts_strogatz')?
%
%   k                 Average number of connections per person
%
%   gamma             Ratio of the number of opinions to the number of people
%                     gamma = (number of people) / (number of opinions)
```

```
%
%   phi                Probability factor with which to choose
%                      the method to be used in each iteration
%
%   iterations         Number of iterations to perform with the simulation
%                      If 0, the function will terminat, when a certain
%                      threshhold for the normed difference between
%                      computation steps is reached
%
%   average_iterations Number of times the whole model will be run
%                      The results will be averaged over these runs
% OPTIONAL
%   max_iterations     Maximum number of iterations to perform when
%                      iterations=0
%
%   threshold          Which normed difference between steps has to be
%                      reached before stopping?
%
%   d_it               Number of iterations performed between checks if
%                      there are significant changes.
%
%   beta               The parameter beta from the Watts and Strogatz
%                      model.
%
% OUTPUT
%
%   mean_distribution Averaged opinion group size distribution
%
%   opinion_dist       Opinion distributions at every thousand iteration
%                      steps. Only calculated if iterations=0.
%
%   mean_number_iterations
%                      Average number of iterations required to fall below
%                      the given threshold.
%
%   step_differences   Matrix with the number of iterations performed so far
%                      in the first row and the respective difference
%                      since the last step (d_it iterations) in the second
%

if nargin < 7, error('Insufficient input arguments.'); end
if nargin < 8, max_iterations = 200000; end
if nargin < 9, threshold = 0.05; end
if nargin < 10, d_it = 10000; end
if nargin < 11, beta = 0.25; end

% If the number of iterations is not specified, we will try to guess
% when to stop iterating, by observing whether the model still
% changes significantly over several iterations (secified by d_it).
if iterations == 0

    % Distributions for comparison
    dist_old = zeros();
    dist_new = zeros();

    % Matrix to aggregate the opinion distributions
    opinion_dist = zeros(n/gamma,1);

    % Number of iterations required to fall below the threshold
    number_iterations = zeros(average_iterations, 1);
else
```

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


% Opinion group distributions for all iterations, will be averaged
distribution = zeros(average_iterations,n);



% Compute 'average_iterations' computations
for i=1:average_iterations
    % Print progress (if more than one run)
    if (average_iterations > 1)
        fprintf('Progress %3.2f%%\n', ((i-1)/average_iterations)*100)
    end

    % Model initialization
    if strcmp(graph,'random')
        people = createRandomSocialGraph(n,k);
    elseif strcmp(graph,'watts_strogatz')
        people = createWattsAndStrogatzModel(n,k,beta);
    else
        error('Unknown graph structure');
    end

    opinions = generateOpinions(n,gamma);

    if iterations ~= 0
        % CONSTANT NUMBER OF ITERATIONS
        % Run model for 'iterations' steps
        [~, opinions] = model(people,opinions,phi,iterations);

        % Compute the opinion distribution with the histogram function 'hist'
        % There are (n/gamma) opinions, so we want to bin the values
        % into (n/gamma) groups
        % We pass a vector with the exact binning points to be used
        opinion_distribution = hist(opinions,1:(n/gamma));

        % Compute the opinion group size distribution
        group_distribution = hist(opinion_distribution,1:n);
    else
        % VARIABLE NUMBER OF ITERATIONS
        % Run the model in steps of 'd_it' iterations
        % and check the difference after each step
        for j=1:(max_iterations/d_it)
            % Continue to run the simulation with the specified parameters
            [people, opinions] = model(people,opinions,phi,d_it);


            % Compute the opinion distribution with the histogram function 'hist'
            % There are (n/gamma) opinions, so we want to bin the values
            % into (n/gamma) groups
            % We pass a vector with the exact binning points to be used
            opinion_distribution = hist(opinions,1:(n/gamma));

            % Add the momentary opinion distribution
            % Only computed for one computation (i.e. the last averaging step)
            if (i==average_iterations)
                opinion_dist(:,j) = opinion_distribution';
            end
```

```
            % Compute the opinion group size distribution
            group_distribution = hist(opinion_distribution,1:n);


            % Compute the normed difference between the last steps
            dist_old = dist_new;
            dist_new = group_distribution;
            diff = norm(dist_new-dist_old)/norm(dist_new);

            % Add the normed difference between the last steps
            % Only computed for one computation (i.e. the last averaging step)
      if (i==average_iterations)
            step_differences(1,j) = j*d_it;
              step_differences(2,j) = diff;
      end

            % Abort if threshhold is reached
            if (diff <= threshold)
                break;
            end
        end
        fprintf('Iterations performed: %d\n',j*d_it);
        number_iterations(i) = j*d_it;
    end

    % Add final group size distribution to distribution matrix
    distribution(i,:) = group_distribution;

end

if (average_iterations > 1)
    fprintf('Progress 100%%\n')
end

% Compute the mean values for the distribution and number of iterations
% if averaging requested
if average_iterations > 1
    mean_distribution = mean(distribution);
    %mean_number_iterations = mean(number_iterations);
else
    mean_distribution = distribution;
    %mean_number_iterations = number_iterations;
end
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 veröffentlichen 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