ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Project Report

# Insert Title Here
...

Name 1 & Name 2

Zurich
May 2008

## 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.

Name 1 Name 2

## Contents

## 1 Abstract

## 2 Individual contributions

## 3 Introduction and Motivations

### 3.1 What is Self-Organized Criticality?

Self-Organized Criticality involves 2 very important concepts: Self-Organization and Critical behavior. These 2 phenomenon arouse in various fields in very different forms.In certain class of systems, the dynamics can be very well explained by a few collective degrees of freedom. This dimensional reduction is termed as "Self Organization". It leads to evolution of global patterns in the system even though all the interactions are local. Let us take an example from the field of physics.Crystallization is the process of formation of crystals. It usually requires a small nucleus for the growth of the crystal. Once the nucleus is formed, crystal can grow on this nucleus. If we consider a particle which is going to crystallize on a given site, the choice of this site is decided only by the local environment seen by the particle. It does not depend on the particle on every other site on the crystal. But this local interaction leads to the formation of an ordered crystalline phase. Bird flocking is an example of self organization from the field of Biology.

On the other hand, there is another set of dynamical systems where the individual degree of freedom keep itself more or less in a stable balance and it is not possible to describe such a system by a few collective degrees of freedom. The key point in such systems is that the interdependence of various degrees of freedom make such systems very susceptible to noise. That's why, these systems can be called as "Critical". Critical state usually represent a point where 2 different phases occur simultaneously at the same time. A critical state has specific temporal and spatial signature. The power spectrum of the quantity representing the dynamics of the system is similar to that flicker noise. (More on flicker noise later!). The spatial signature of a critical state is "self-similarity" which is what we find for the case of fractals.

Self-Organized criticality combines the above 2 concepts. In it, the dynamics of a physical system converge to a critical point independent of the bestowed boundary conditions. This critical point commonly is not the equilibrium point in a traditional, physical sense. We usually say that the *critical point* is an attractor of the system dynamics. The concept is best understood by example, so we shall bring up a very educative one introduced first by [1]. Imagine a sand pile in 2 dimensions. One can discretisze this pile into cells, which are denoted with the indices $n$. Each of those cells has an amount of sand grains on it, which is directly corresponding to the height $z_n$. Now we state the simple rule, that, if the difference in height between two neighboring cells becomes larger than a predefined limit, the higher cell will give grains to the lower cell (avalanche). Now every timestep, we drop a grain on cell one. Independent of the initial conditions, eventually, a straight slope will evolve. The critical point is reached if every difference between two neighboring cells is exactly the limit. What happens if we drop the next grain? Then the difference $z_1 - z_2 > p$ and one grain will drop on $z_2$. Subsequently, the difference $z_2 - z_3$ will be too big and the process repeats until it hits the last cell. We therefore call a state critical if a yet so small disturbance can propagate throughout the whole system. It is important to note here, that this critical state is *not* the equilibrium state of the system, since that would be a flat surface.

## 3.2 Forest-Fire Model (FFM) as an example of Self-Organized Criticality

A Forest Fire Model is basically a cellular automata. Each cell can have one of the following three states:

- 0: A green tree
- 1: Empty site
- 2: Fire

The dynamics of the system is defined though the following rules.They are applied to every step as we go from 1 time step to the next.

1. If a tree is burning at a given time step (2), we make it an empty site during the next time step.
2. If a site is empty (1), a tree can be grown on it with a probability p. This means that if we found out a site to be empty, we generate a random number between 0 and 1. If the number is less than p, we change the state of the cell to 0, otherwise it stays an empty site.
3. If a site is a green tree (0), we burn this tree in the next step if:
    - Either of its neighbors is a burning tree.
    - We generate a random number between 0 and 1, if it is less than f (lightening probability), we burn this tree.

It is important to note that the Forest Fire Model does not only apply to forest fires as its name would suggest. A whole class of problems follows the same basic rules and can be treated accordingly. An example would be the spreading of a disease, where one can directly transform the above rules and states.

This model describes an SOC state in the regime where the ratio of p and f is very large and p itself is very small. An important question is how to characterize a state as an SOC state. This is described below.

### 3.3 Motivation to study Forest Fire Model

Why is it important to study Forest Fire Model? Forest fire models apply to a wide range of problems and are therefore helpful in predicting the behavior of such systems. Self-Organized criticality is the concept that describes the dynamics of such models and the two fields are therefore closely connected. One has closely studied these effects to make better predictions about forest fires. The main problem was, that in most cases, even relatively small fires were extinguished by the authorities. Since that led to an *overcritical state*, the probability for a huge, devastating fire rose quickly. This has happened in the past and had a serious impact on the ecosystem of those areas. Since the problem of self-organized criticality has been understood, one has stopped to extinguish every little fire, therefore avoiding to reach an overcritical state at which the disturbance (a small fire) can propagate throughout the whole system (large-scale fire). The same ideas can be applied to a variety of problems such as disease spreading or urban planning.

### 3.4 What do we want to know from the Forest Fore Model- The QUESTIONS

First, we would like to understand what are the quantitative characteristics of a Self Organized Critical state in the Forest-Fire model? We would like to have a well defined idea regarding when does the Forest Fire model manifest SOC behavior and why. We would like to understand what are the various phases seen in a Forest Fire model besides the Self Organized Critical phase. It will be important to know what are the parameters characterizing the phase transition in the Forest Fire Model. Some of the ambitious questions will be to understand the dependence of SOC of the Forest Fire model on the dimensionality of the Lattice or on the neighborhood selection on the grid.// It will also be interesting to find out if computational restrictions like finite size grids, periodic boundary conditions and cell updating procedures have a quantitative effect on the behavior of the model?

## 4 Description of the Model

### 4.1 Understanding the SOC regime of Forest Fire Model

For FFM to be an SOC, we need to ensure 2 things. One is that the probability of growing a tree should be much larger than the probability of lightening. We can intuitively think about the ratio of p and f as the number of trees growing between 2 lightening strokes.This condition ensures long range correlation in the system (which will lead to criticality) building out of purely local interactions (randomly growing trees). Another condition which we need to ensure is that the time needed to burn a large cluster should be much smaller than the time needed to grow a tree. This condition ensures invariance under a change of the length scale [2]. The 2 conditions can be written together as:

$$T_{burn}(cluster_{max}) << p^{-1} << f^{-1} \tag{1}$$

### 4.2 Cellular automata

A cellular automata is a very powerful way of simulating problems which are defined by a set of rules. It is usually simulated on a grid, but not necessarily restricted to those geometrical constraints. The main characteristics are:

- Every grid point has a state
- Grid points change their state depending on the neighbor states according to a set of rules
- Random actions may be introduced

In the example of the FFM, every grid point has three states (empty, alive and burning) and four rules, the most obvious of which is change state to burning if you are alive and your neighbor is burning.

Cellular Automata can exhibit very complicated behavior when fed with very simple rules, which is the main reason why they are so powerful. There are similarities to agent-based models and networks, but it is not to be mistaken for one of these.

The second approach to grid-based updating is with a second grid that stores temporary information. [Insert Description here]

### 4.3 Critical Exponents

Usually, a critical point is associated with certain power law distributions of various physical quantities characterizing a critical state. Power laws are functions of the form $P(s) \propto s^{-\tau}$ . These functions are found in many data sets, such as the size distribution of cities in a certain area. Take any country, one will find one very large metropolis, a few large cities, many medium sized towns and a huge lot of small towns. Power laws have also prominently been found in data sets of earthquakes, where they apply almost perfectly, meaning that small earthquakes happen very often in respect to the frequency of big ones. Power laws imply that the frequency of a general data point is inversely proportional to its magnitude.

At a critical point, the power laws are characterized by critical exponents which are the indicators of the critical state. They usually depend only on the dimension of the system and the nature of interactions/ dynamics involved. These exponents are independent of the initial condition and the size of the system. These critical exponents can be calculated either by mean field approaches or through numerical simulations. The mean field approach becomes more and more exact in higher and higher dimensions.

In context to the FFM, it is proposed as a way of quantifying the frequency of Fires with respect to their cluster size.

## 4.4 Key Parameters in FFM SOC

To find out whether a basic FFM exhibits SOC behavior, we need to define a set of parameters which serves as indicators for the desired analysis.

- The time needed to burn down a forest cluster: This is heavily dependent on the form of the implementation. If we choose to implement it in a way like [reference needed], we will have an instantaneous fire which essentially takes no time to burn and just resets all the connected trees. If however, we choose the form of a visible fire, meaning that the ratio $f/p \ll 1$ is not 0 in limit and that it takes the fire one time step to advance a grid cell, then the time needed to burn a forest cluster is an important variable we need to extract from the simulation.

- Number distribution of the size of clusters: Here we first need to define what a cluster is. A cluster is a set of neighboring cells (Von Neumann neighborhood) all obtaining the same state. When we talk about forest clusters, we of course mean connected trees. In the implementation where the fire spreads infinitely fast, the forest cluster containing the igniter cell is the same as the burnt area. The size distribution therefore is a very good indicator for the fire size distribution.

- The mean number of forest clusters in a unit volume $n(s)$ , where s is the number of trees in a cluster.

- The radius of a cluster, it is defined as the root mean squared distance of all the elements of a cluster from the mean. It gives an estimate of the size of cluster.
- Number of fires per unit time step: This is a tuning parameter of the model.
- Correlation length.

We have tried to implement the model in a way that the measurement of these desired variables is possible with the least needed effort.

## 5 Implementation

As discussed before, there are two main implementation methods for the model: One way is to update the whole grid every time step, whereas the second method picks a cell randomly at every time step an only updates the chosen one. One of us worked on the first method and the other one worked on the second method.

### 5.1 Grid Based Updating

The algorithm for grid initialization looks like:

```
Grid(N,N)=zeros(N,N)
rand_grid=rand(N,N) % we generate a grid of random numbers
m=1;
while m<=size(Grid,1) % Loop over all y-coordinates
n=1;
while n<=size(Grid,2) % Loop over all x-coordinates
if rand_grid(m,n)<k %If the random number for a site is less than k,
%we make the site empty, else it has a tree
Grid(j,k)=1    % chossing different k will lead to different
%initial state of the grid
end
end
end
```

Basically, we initialize the grid with trees and empty sites distributed randomly on the grid. The updating of the grid based on the simple rules described above can be done in 2 different ways. One is when we make a copy of the current grid ($Grid_{copy}$) and then update the current grid using this $grid_{c}opy$. The other is that we use the current grid (which is being updated) as the one which also responsible for updating. The 2nd procedure require us to pick the sites randomly and update them, but it will be hard to keep track about the sites which have been updated at a time step. In our simulations, we follow the first procedure whose algorithm is described below:

```
it=1:
while it<=Iter % Loop over all time steps
    Forest_grid_temp=Forest_grid;  %Forest_grid_temp is the copy
    %which is used to update the grid at the next time step.
% we need to take care that the neighbors of every cell are correct
%as based on the periodic boundary condition.

%This has been handled by making the size of the copy as 2 units
%larger in each of the dimensions

    Copy the first column of Forest_grid to last column of Forest_grid_copy
    Copy the last column of Forest_grid to first column of Forest_grid_copy
    Copy the first row of Forest_grid to last row of Forest_grid_copy
    Copy the last row of Forest_grid to first row of Forest_grid_copy
    m=2;  %looping over the Forest_grid and its copy,
    while m<N
        n=2;
        while n<N
            if Forest_grid_temp(m,n)==2
                Forest_grid(m,n)=1;
              %a fire becomes an empty site

            elseif Forest_grid_temp(m,n)==1
                emp=emp+1;
                if rand(1,1)<p
                    Forest_grid(m,n)=0;
                end
              %an empty site becomes a tree if a generated random no. is <p
            else
                if (rand(1,1)<f || Forest_grid_temp(m-1,n)==2 ||
                Forest_grid_temp(m+1,n)==2 || Forest_grid_temp(m,n-1)==2 ||
                 Forest_grid_temp(m,n+1)==2)
                     Forest_grid(m,n)=2;
                %if any of the neighbours of the current site is burning or a
                % random number is generated which is greater than lightening
                %probability, we burn the tree.
                end
            end
            n=n+1;
        end
        m=m+1;
    end
    it=it+1
end
```

One of the important reasons why we made another model is that the algorithm is quite slow because of all the loops and if-statements,we are using.

The next important thing which we need to find out is indexing uniquely all the clusters of trees. The algorithm is described below:

```
Index=3;
m=1;  %looping over the Forest_grid and its copy,
while m<N
   n=1;
   while n<N
    if Forest_grid(m,n) is an un-indexed tree
    we search all its neighbors to find the smallest index
    if index is found,
    Forest_grid(m,n)=index;
    We also update the index of all the nearest
    trees with this index
    if index is not found,
    Forest_grid(m,n)=Index;
    We also update the index of all the nearest
    trees with this Index
    Index=Index+1;
    end
end
end
end
%The above loop will be sufficient if we have absorbing boundary conditions.
%Periodic boundary condition requires that we should check all the indexing
%again
m=1;  %looping over the Forest_grid and its copy,
while m<N
   n=1;
   while n<N
```

```
    if Forest_grid(m,n) is a tree
    we search all its neighbors to find the smallest index
    If the index of any of the neighbors or the Forest_grid(m,n)
    is different than this smallest index, we replace all the trees
    of that index with this smallest index
end
end
end
```

Once the indexing is done, we can simply count the number of trees corresponding to a given index and can find out the number of number of clusters of a given size (where by size, we mean the number of trees in the cluster).

Calculation of the radius of a cluster is little subtle. Naively, it just the root-mean squared distance of all the elements from the cluster mean. The formula can be simplified as:

$$R = \sqrt{\frac{1}{N}\sum x_i^2 - \frac{1}{N}(\sum x_i)^2} \tag{2}$$

This would have worked well if we have absorbing boundary conditions, but since our boundary is periodic; there can be possible a cluster which is going over the boundary. In that case, above formula can lead to a cluster mean which is outside the cluster. In such a situation, we need to rotate our grid to calculate the cluster radius properly and efficiently. The underlying idea is very simple. If you are scanning through different columns (rows) from left to right (top to bottom), you should get an element of a cluster in every column since the starting column for that cluster. In case, there is a break, it implies that you have cluster going though the edges, otherwise it cant be a cluster. Another important property of a cluster going through their boundary is that it should have a member in the leftmost (topmost) and rightmost (down most) column (row). We exploit these in the following algorithm:

```
%Column shift
define variables F and ml for every cluster index
initialize F and ml with zero for every cluster
m=1
while m<=N  %going through all the columns
n=1;
while n<=N  %going through all the rows of a given column
if the current element is a tree,store the index of current
element in variable cl
```

```
if F(cl)==0
%this is the first column where an element of this cluster appears.
if m~=1
F(cl)=3;
%this implies that this cluster cannot be connected via the boundary
else
F(cl)=1;
ml(cl)=m;
  end
elseif F(cl)==1
if m~=ml(cl)+1
%this is the case when we have atleast one column in between where no
%element of the given cluster is found and this can happen only if
%the cluster passes through the boundary.
  F(cl)=2;
end
ml(cl)=m;
end
end
end
%The column shift needed for a given cluster can be calculated as:
if F(cl)==2
shift_col(cl)=N+1-ml(cl)
else
no shift
end

%An exactly similar procedure is followed to calculate the row shift for every cluste
%index where we go though all the columns for a gicen row and very all the rows.
```

### 5.2 Random Based Updating

In a random based updating implementation, at the beginning of every time step, one random cell is chosen. It then checks for the rules of the model and acts accordingly. In order to see fires, we need to have instantaneous burning, which is actually permitted by the model. However, it changes the dynamics of the system drastically. But more about that later. The following steps are taken for each timestep:

1. Choose a single cell

2. If cell is empty, grow a tree with probability $p$
3. If cell is a tree, with probabiltiy $f$, burn it and the connected cluster

In Matlab, this looks something like this:

```
for i=1:t % Loop over all timesteps
(j,k)=ceil(Grid_Size*rand(1,2)); % Choose a random cell
if  Grid(j,k)==0 % If the cell is empty
\\With some probability p set G(j,k)==1;
end
if Grid(j,k)==1 % If the cell is alive
\\with some probability f execute burn(j,k);
end
end
```

This implementation of the forest fire model has several advantages:

- There are no problems due to updating processes
- Much shorter implementation
- Easier to extract data

The last point is to be explained more extensively: Every time a tree ignites, the whole connected cluster is burned down. Since the whole process happens in a single timestep, we were able to use a recursive function. This function would search every neighboring cell and pass the function on to it if it is alive, just after burning itself. Now with this function structure, we were easily able to retrieve much wanted data such as total function calls (Fire size) or recursion depth (Cluster size). Problems with this implementation will be discussed later.

### 5.2.1 The "burn" function

The burn function makes the magic happen for the whole program. It basically takes the current grid cell coordinates and the recursion depth as input. Here is what it does with that:

1. Set the current cell as 0 (empty)
2. Add one to the recursion level
3. Retrieve the direct neighbor coordinates from the *neighbor* function

4. check every neighbor for being alive. If so, call the *burn* function with the neighbor coordinates.
5. Add the returned values to the current values (trees burned)
6. If no neighbor is alive, return trees burned as one.

In other words: This function propagates throughout the system until it reaches a grid cell which has no alive neighbors. It then goes back to the last grid cell where it did have another possibility to go to and chooses that one. This process repeats until there are no trees left in the cluster. Each time the function goes back on its path, it accumulates all the trees burned by that branch. In the end, that gives us a very clear picture about how many trees were burned by evaluating the initial call.

**Problems with the *burn* function** The big problem with this function was, that it would exceed the available stack memory pretty fast. This would always happen if the grid is relatively full. What then happens is that there is no boundary to the cluster, since we are using periodic boundary conditions. The function would propagate throughout the whole system in one single branch. This causes *MATLAB* to crash gracefully and tell the user that one should set the recursion limit to a higher value. What we learned from that, is, that a normal computer crashes at about 2'500 recursions. Now this means that we would not be able to simulate grids of size $25 \times 25$ or more which is not acceptable.

**Solution** The solution obviously was to limit the recursion depth of the function. This was relatively easy to implement, we just had to add 1 every time the function was called. There was, however, a payoff. Since the Function did not have "full" freedom anymore, it could happen that it did not burn the whole cluster. The fraction was in the range of 0.99, but still, it was not quite the optimum.

**Improvements** were made in these ways:

- The order in which the neighbors were checked was turned into random. This effectively resulted in the "drunkards walk" and proved to be a little more effective.
- The recursion depth was discretized into directions, which made it also more effective.

### 5.3 Diagnostics

Since we wanted to determine some properties of the simulation, we had to implement diagnostics. These were the properties measured:

**Number of alive trees** The Number of trees alive is relatively easy to measure in the random based updating implementation. This comes from the property that there are only two states, 0 (empty) and 1 (alive). It therefore is sufficient to execute

```
N=sum(sum(Grid));
```

This value also allows us to compute other properties very quickly, like the mean number of alive trees in a simulation or the number of empty sites. In the grid based updating implementation, this was implemented with a loop and a counting variable. The Loop would simply check every entry of the grid for being 1 (alive) and subsequently add 1 to the counting variable.

**Trees burned in a fire** This was quite difficult to measure in the grid updating version. The big problem was that there was a possibility of multiple fires happening at the same time. It is therefore not possible to just loop over the whole grid and count the burning trees. One needs to track single fires and only add the trees burned by this incident. The other big problem is, that there is also the possibility of trees growing at the edge of the cluster that is on fire. This means that it is not possible to just search for the entire connected cluster, since its size can (and does!) vary during the fire. In the random updating version, this was a lot easier, since the fires would spread instantaneously. Since we were using a recursive function to burn the cluster, we could just pass the trees burned by each sub-branch of the function back to the branch head (initial function call). There are also no issues of multiple fires happening simultaneously and changing cluster sizes.

**Cluster size distribution** For long simulation times, the cluster size distribution approaches the fire size distribution if the fires spread instantaneously.

## 6 Simulation Results and Discussion

The first step in the simulation was to see if we could reproduce some of the results found in the references. For starters, we wanted to see if the Formula $\overline{s} \propto \Theta^{-1}$ would hold. This shold be explained: We define the factor $\frac{f}{p}$ as $\Theta$ and the mean number of trees burned by one fire as $\overline{s}$. Then the change in the Number of trees present in the system for some large time t is given as

$$\Delta N = p \cdot t - f \cdot \overline{s} \cdot t$$

If we reach a critical point, then this difference should be zero for large times. Then we can write

$$p \cdot t = f \cdot \overline{s} \cdot t \text{ or } p = f \cdot \overline{s}$$

Since we defined $\Theta = f/p$, we can now write

$$\overline{s} \approx \Theta^{-1}$$

What does it tell us if we find this to be true? It tells us, that the system dynamics have a stationary point. However, since we average over quite a large time, the result is not a strong one, but merely a check if the simulations work right. In a first attempt to recreate this result, it did not work. We plotted the curve $\overline{s} \cdot \Theta$ for values of very small to very large $\Theta$ (which shold be constant), and observed a small, but relevant slope with quite big differences for small values of $\Theta$. Now there comes a problem with the implementation. Since we only plant a tree with probability $p$ if the site is *empty* and only set a site on fire if the site has an alive tree (and no actions are taken if these requirements are not fulfilled), the above equation has to be rewritten in terms of the probabilities:

$$\Delta N = \frac{(G - N)}{G} \cdot p \cdot t - \frac{N}{G} \cdot f \cdot \overline{s} \cdot t$$

where $G$ is the total Number of Grid Points. For a stationary point, this $\Delta N$ again has to be 0, therefore we can write

$$(G - N) \cdot p = N \cdot f \cdot \overline{s}$$

This means that

$$\Theta \cdot \overline{s} = \frac{G - N}{N}$$

should hold. Now $\frac{G-N}{N}$ is indeed a constant, which should not change. However, it is implicitly dependant on $\Theta$, since this has a direct impact on the mean number $N$ of alive trees in the system. Imagine for example a very small $f$, meaning that fires are very rare. Then this constant factor is very small because $N$ approaches $G$. For larger values of $\Theta$, $N$ will become smaller and therefore, the factor $\frac{G-N}{N}$ will become larger. That was the result that we obtained from the simulation. It therefore suggests that the simulation is working as expected.

## 7 Summary and Outlook

## References

[1] Per Bak, Chao Tang, and Kurt Wiesenfeld. Self-organized criticality. *Phys. Rev. A*, 38:364–374, Jul 1988.

[2] Siegfried Clar, Barbara Drossel, and Franz Schwabl. Forest fires and other examples of self-organized criticality. *Journal of Physics: Condensed Matter*, 8(37):6803, 1996.