Homework 3.(b)

Modeling Complex Systems, Javier Lobato

Due date: Thursday, March 29, 2018

A CA model and differential equation comparison

The basic mechanisms of the system can be modeled in either the CA model and the SIR differential equations model. However, group behavior and heterogeneous conducts can't be modeled with a differential equation model. The disease will never die out with one model of this kind. This can be explained due to some randomness associated with the CA model - which does not exists in the differential equations. Also, the discrete changes in states are more easily represented with a CA.

Focusing on the CA model, the deterministic and synchronous version differs with a stochastic and asynchronous version in the predictability. When a probability or a random number is included into a set of equations, these will have a more 'real' behavior because the system will not behave always in the same way.

Finally, both CA and differential equations model have their pros and cons. CA should be used to model the type of group behaviors with emergent properties that can't be analyzed in a model with differential equations.

B Fire spreading model

Motivation and research

The exposition given by lecturer Hbert-Dufresne during the previous days motivated my research on the topic of forest fire prediction and modeling. There are a lot of different papers and unanswered questions about the topic. The question I came with was

'How do the number of fire focus affect the speed of the fire front and the forest total burning?'

Doing some research, one can find different models of cellular automata to analyze fire spreading. Choosing a simple model and making some changes allowed to explore the different possible cases and scenarios.

Why can't we use differential equations?

There are different reasons why a differential equation modeling will be worse to analyze this case. The first and most obvious one is the discrete changes in state: in the fire spreading only discrete states will be analyzed. Each cell may be empty, vegetation, fire or burnt. This discrete states will be hard to implement in a differential equation model. Another reason is the heterogeneity of individuals behavior: each state will be described by a set of different rules, modeled with some probability that will vary the future state of the cell depending on a random number. Finally, the cellular automata model has no numerical error associated with the approximation and round-off errors of the numerical method used. Thus, in a CA only the modeling error will be the predominant one.

Experimental design

The cellular automata model is executed over a square grid with Moore neighborhoods (as proposed in Almeida, R. M., Macau, E. E. (2011). *Stochastic cellular automata model for wildland fire spread dynamics*. In Journal of Physics: Conference Series - Vol. 285, No. 1, p. 012038). The boundary conditions are not discussed in the paper given that for any of their cases, the fire never reaches the limit of the grid. For the case created, both toroidal and absorbing boundary conditions are implemented: this way different configurations can be explored. Also synchronous and asynchronous updates of the map, having better and more realistic results with the asynchronous model - what makes sense given that fire does not follow an order. In this paper, the basic CA consists of 4 possible states: empty (E), vegetation (V), fire (F) and burnt (O). The states are implemented with numbers to make easier the storage in arrays, having that 0 corresponds to the empty cell, 1 to the vegetation, 2 to the fire cell and 3 to a burnt cell. The rules that model the behavior depend on the value of three parameters, that go from 0 to 1 and describe the probability of some events:

- D: this parameter is only used on the creation of the initial map. If $D = 0$ the map will be completely empty and if $D = 1$ the map will be covered with vegetation (making easier to the fire to spread). The map is first created with `zeros`, having a full empty forest. Then the map is looped, adding a vegetation cell depending on the probability D.
- I: the possibility of a vegetation cell to become a fire cell. If the value of I is very low, there is a high possibility that the fire dies out. Even with high values $\neq 1$, there is a small possibility that the fire doesn't expand - having values that perturb the mean and standard deviation.
- B: this probability expresses the change of a fire cell into a burnt cell. If it is $= 0$, fire cells will never turn into burnt cells (having a never-ceasing fire). On the other hand, if $B = 0$, the fire cells will immediately become burnt.

Although the rules may be inferred from the probability definitions, they are going to be shown now:

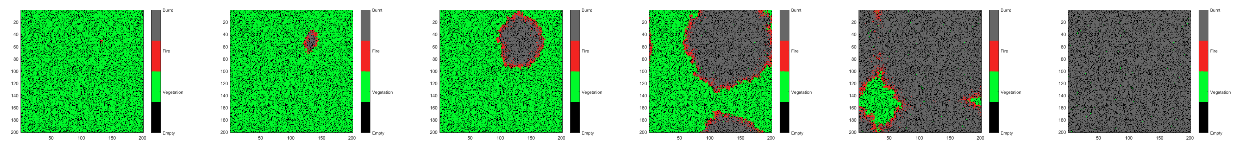
- An empty cell will always stay as an empty cell
- A vegetation cell will have a probability $1 - (1 - I)^{Fire\ Neighbors}$ of becoming a fire cell. This possibility depends on the number of neighbors that are also in fire
- A fire cell will have a probability of B of becoming a burnt cell. If the probability value is not reached, the cell will remain as a fire cell
- A burnt cell will stay like that with no possible change

While the rules may seem very straightforward, the results obtained are impressive.

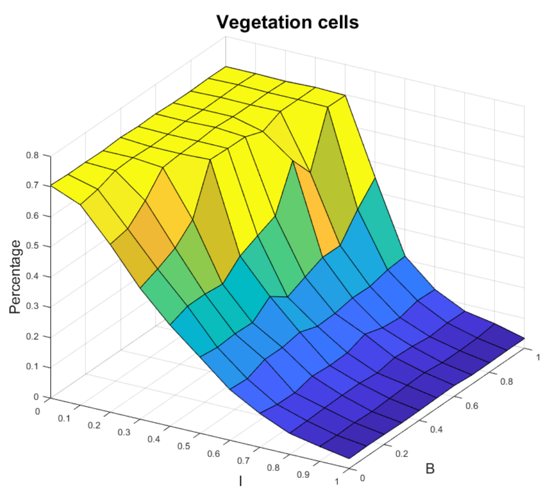
Results

The main objective is to see what happen when there are more than one fire seats in the same forest fire. When the fire is produced by a pyromaniac (or by a group of them) there are different points where a fire starts. How the number of fire seats affects the behavior and expansion of the wood fire is the question analyzed, searching for methods to extinguish. In order to test the different number of fire seats, the initial empty-vegetation map should be 'perturbed' with a fire cell (otherwise, fire will never begin and expand). The number of fire seats is another input of the system, having a random position for each one of the desired fire focus.

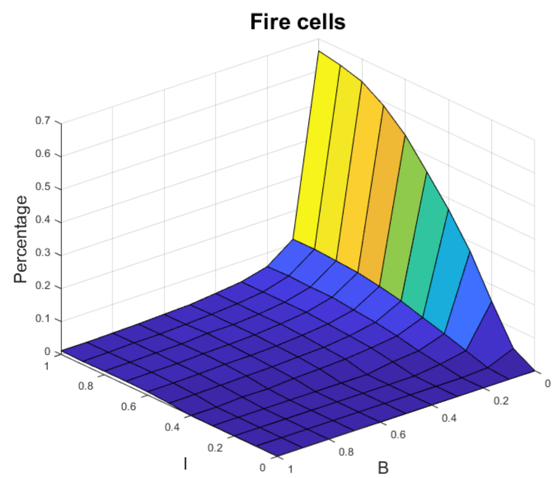
After having tested both types of boundary conditions, the main disadvantage of the absorbing BC is that the time spent to burn the whole forest depends on the position of the different fire focus (which is randomly determined). Thus, the toroidal boundary condition will be mostly used - having always the 'same' area to burn. The next figure shows different steps of a sample simulation run:



First of all, a parameter sweep was done in order to see if there were any transition and the general behavior of the system. Varying G will only change the initial map, having more or less dense forests. Thus, probabilities I and B will be swept from 0 to 1, having a 3D map. The outputs of the function are the number of each cell state for each time step, the location of the fire seats and the time to have the full forest burnt. Analyzing the percentage of each cell state (over 1) for the last time steps, the results are:



Vegetation cells



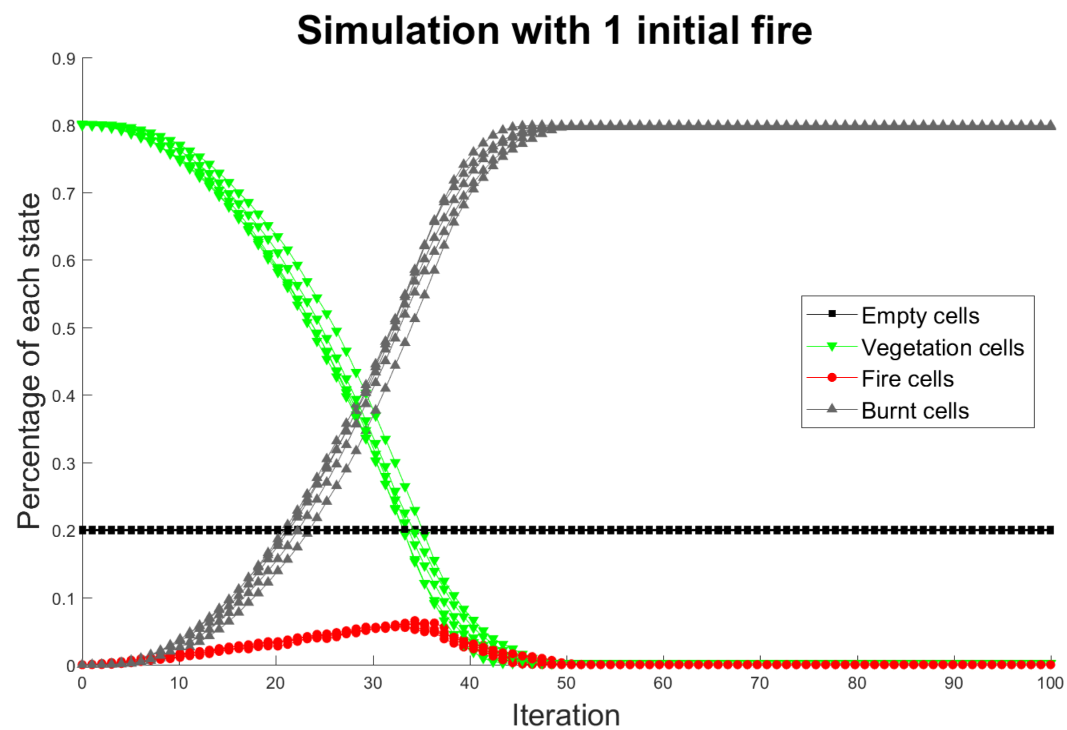
Fire cells

Parameter sweep of $B, I \in (0, 1)$ done with $D = 0.7$ and just 1 fire focus, in a 201×201 grid.

It can be seen that the system modeled with the CA is very smooth and that the only big step is seen when $B = 0$, having that all fire remains burning forever. There is also a step on the vegetation cells for some values of B and I, when there are a lot of probabilities that the fire doesn't expand.

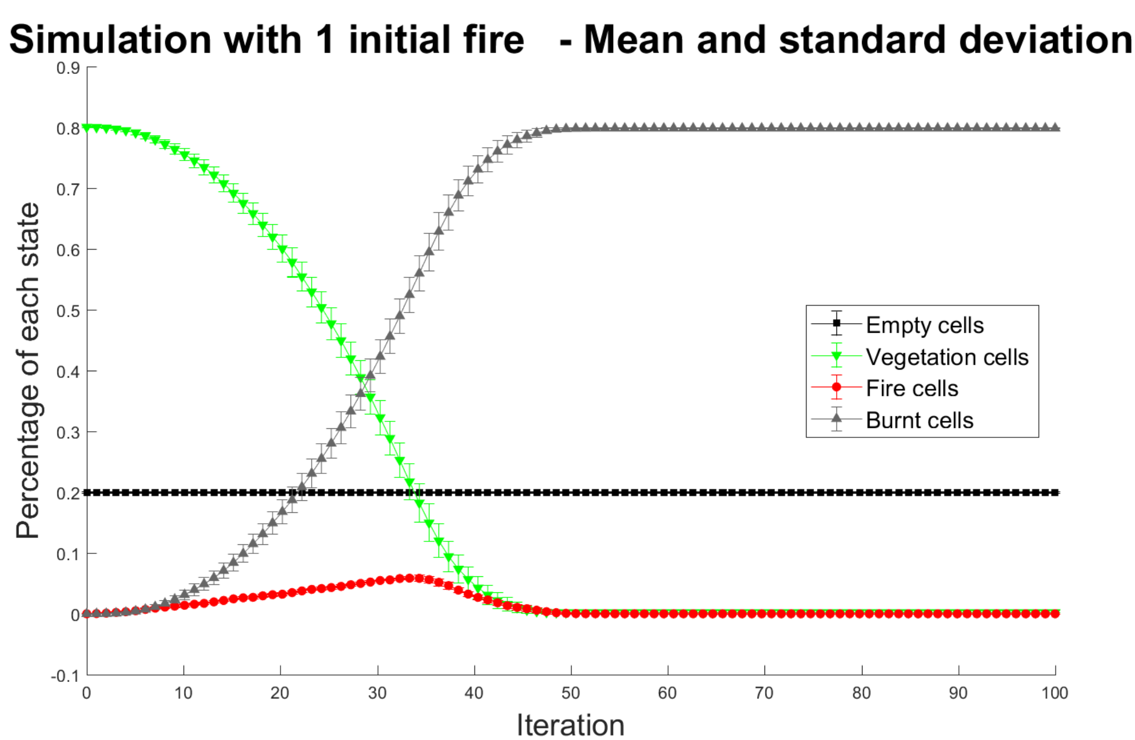
Once the parameter sweep has been done, some basic parameters will be chosen to perform the rest of the calculations. The different probabilities that will be used are $G = 0.8$, $B = 0.6$, and $I = 0.7$, because they give the better and most accurate results.

Although a firewall initial configuration was implemented and tested, the potential of the code resides on the capacity of emulating different fire focus. With the parameters said and a 201×201 grid, an asynchronous update, and toroidal boundary conditions, a sweep of initial fire focus were simulated. A range from 0 to 20 fire seats was tested 5 different times to allow randomness to occur and to see how disperse the results are. For example, the simulations for 1 initial fire gave the next results:



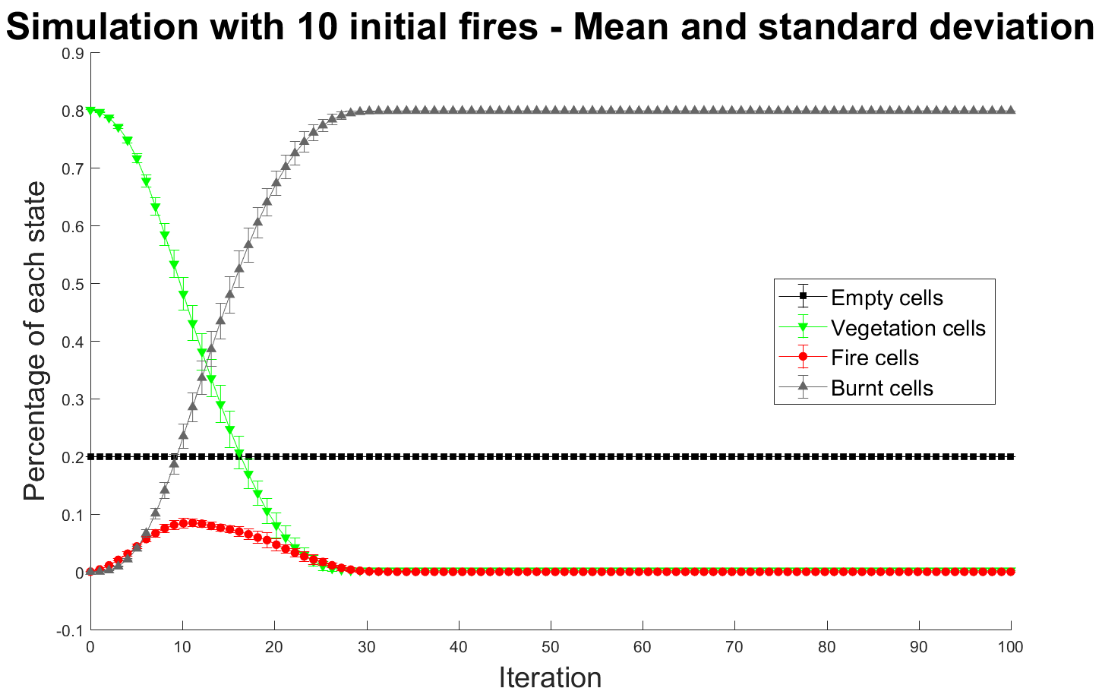
1 initial fire: percentage of each state plotted for each simulation

In order to better see the results, the mean and the standard deviation were computed and plotted, having the next figure:



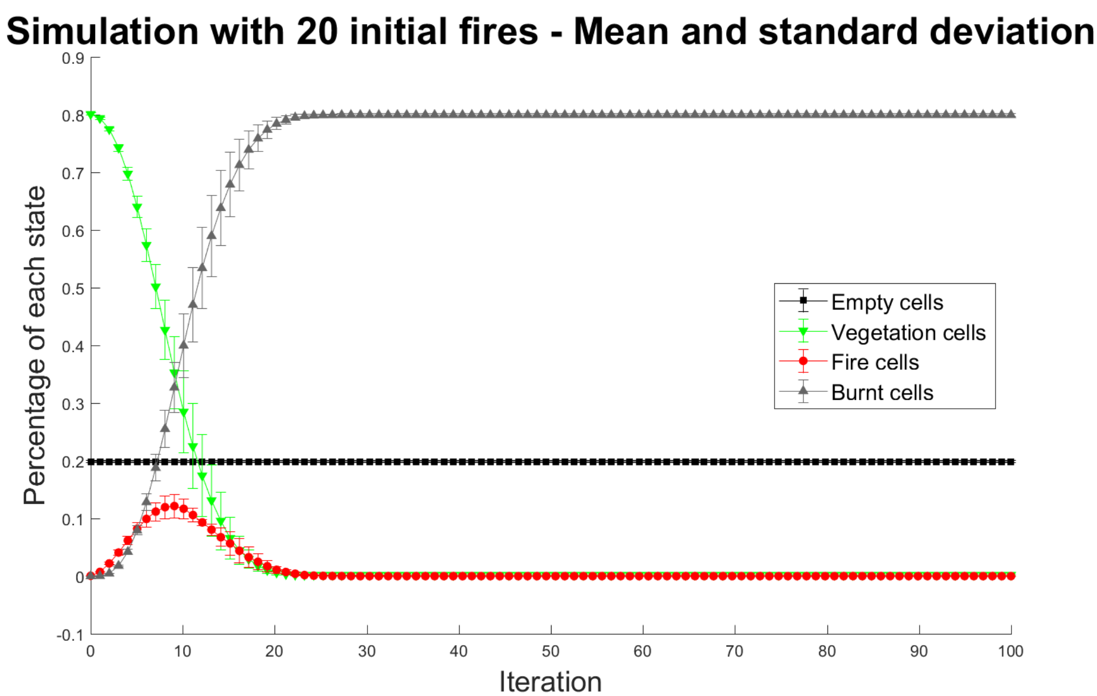
1 initial fire: μ and σ of the percentage of each state averaged for each run

It can be seen that there is not a very big variation for each one of the runs with the same set of parameters. Let's analyze bigger numbers of initial fire seats:



10 initial fires: μ and σ of the percentage of each state averaged for each run

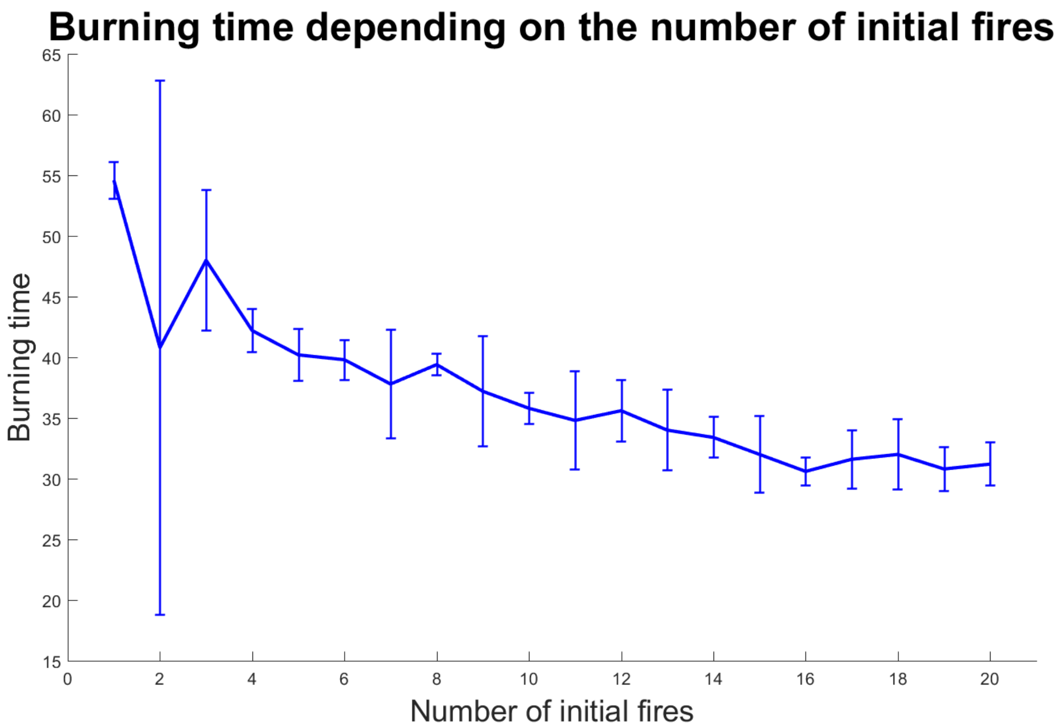
It can be seen that the number of iterations that it takes to burn the vast majority of the vegetation (there is always some small gap where fire can't enter) is way smaller when there are 10 initial fires than when it was just one. Let's analyze the case with 20 initial seats of fire:



20 initial fires: μ and σ of the percentage of each state averaged for each run

In this latter case, the standard deviation is bigger, having as before a smaller amount of time required to burn the most part of the forest.

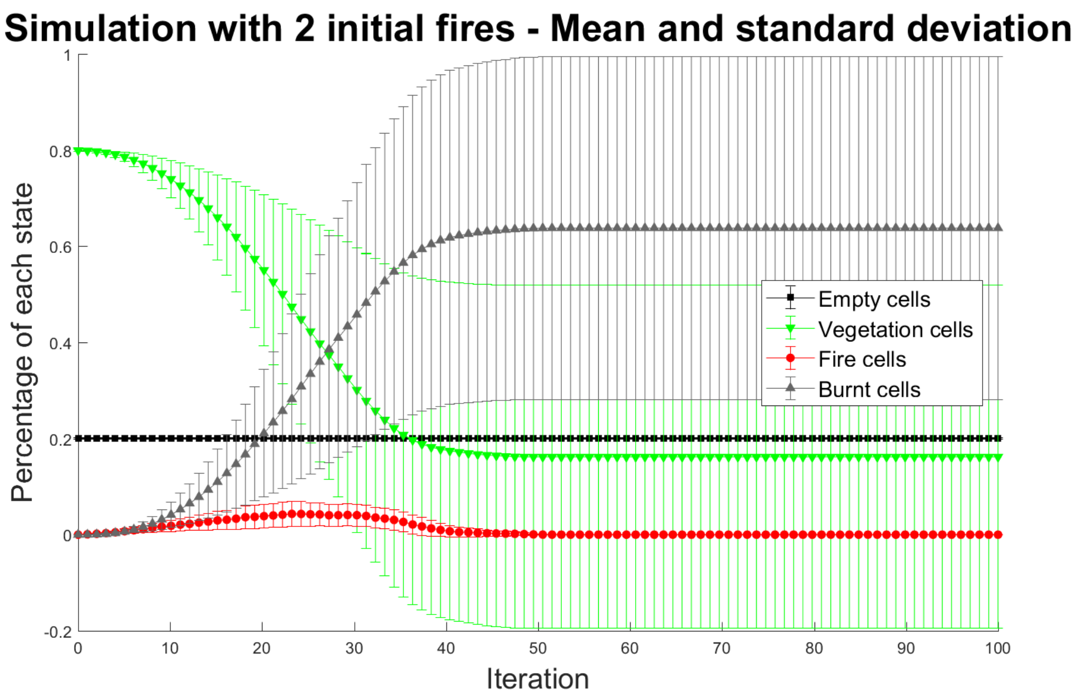
To know if time will eventually approach an asymptotic behavior, let's plot the time that it takes to burn the whole grid versus the number of initial fires:



Number of fires with the μ and σ of the burning time

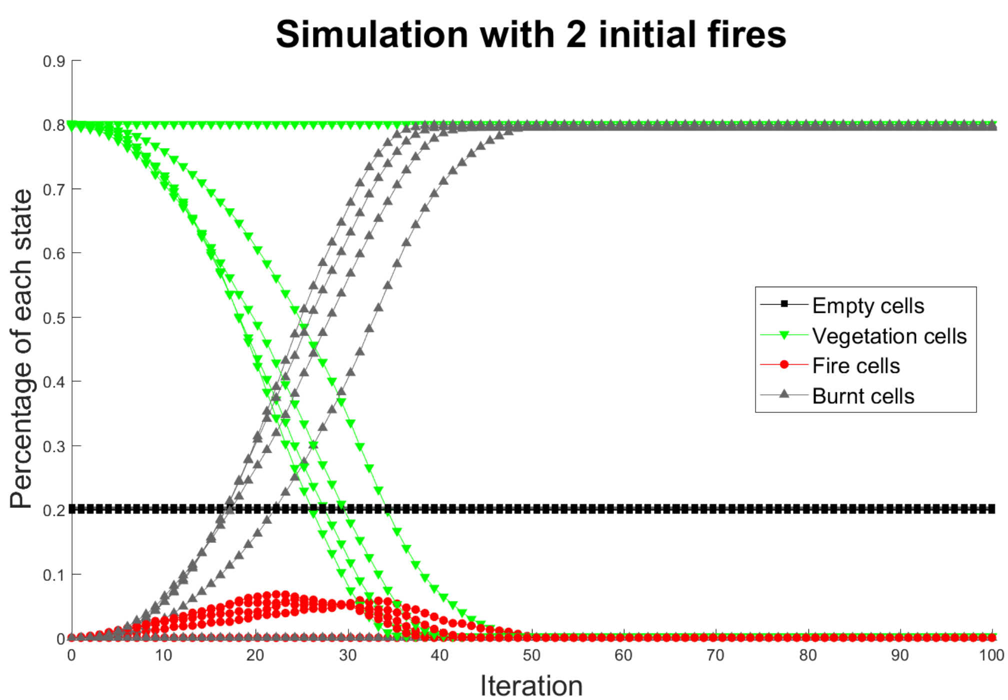
It can be seen that there is a kind of asymptotic behavior when the number of initial fires is higher than 14. The value when the number of initial fires is 2 looks weird compared with the general trend of the system.

Analyzing the plot of μ and σ for the case of two initial fires:



2 initial fires: μ and σ of the percentage of each state averaged for each run

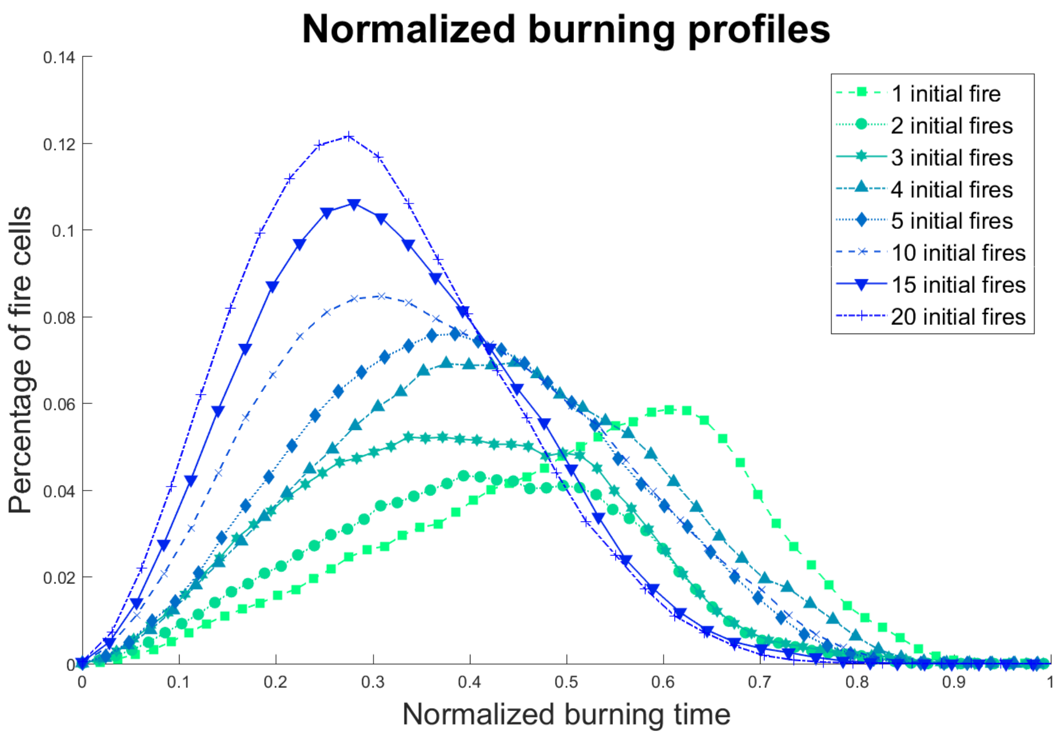
The result here shown that there is a big variation with respect to the mean values, having that the mean values don't go to 0 (vegetation) and 1 (empty) as seen for previous cases. Taking a look at the figure that shows the value of each individual run of the same 2 initial fire case:



2 initial fires: percentage of each state plotted for each simulation

It can be clearly seen that there is one case when fire extinguish before expanding, having that the forest hasn't been burnt and the vegetation remains as it was.

Another thing that it can be noticed at the naked eye is that the higher the number of initial fires, the smaller the time it takes to burn the whole grid. Analyzing closer the burning rate, depending on the initial fires, there will be a burning profile. Normalizing those with respect to the total burning time (having that the graph goes from 0 to 1).



Time versus intensity of the normalized burning profiles

The initial slope is steeper with a higher number of initial fires. Also, the peak is reached before - given that in less time, the same amount of vegetation will be burnt. On the other hand, the decreasing slope that goes once the maximum is reached is very similar for all cases. For further developments, the distance between the different fire focus is a relevant factor when analyzing the time spread of the fire and it should also be considered. Analyzing it could give a broader view on how to cut off this problem.

Given these results, it can be stated that the higher the number of initial fires, the fastest the forest will burn. It can also be said that with a higher enough number of initial fires, the time of burning a determined are will not decrease anymore (or at least that is what has been possible to evaluate).

Used code

`fireSpread.m` file

```
function [EVFO, fip, finalT] = fireSpread(dim, D, B, I, maxt, initializationConf, BC, sync_async)
% fireSpread(dim, a, g, maxt, IC, infectionProb, syncOpt, immgrtRate)
% Implement fire spreading model  with a cellular automata
% based on: R. M. Almeida and E. E. Macau "Stochastic cellular automata
% model for wildland fire spread dynamics", J. Phys.: Conf. Ser., 2011
%
% MANDATORY INPUTS:
% dim: dimensions of the square map (same dimension for both sides)
% D: probability that represents the heterogeneity of vegetation (D
%       represents full vegetation and (1-D) is an empty forest)
% B: fire length of a cell to go from burning to burnt
% I: ignition probability of a vegetation cell near a burning one
% maxt: maximum number of timesteps to run (user can abort early)
% initializationConf: the possible initial cases are:
%       - vertical firewall
%       - horizontal firewall
%       - fire focus number
% BC: two possible types: 'absorbing' and 'toroidal'
% sync_async: two possible types of updating: 'synchronous' and
%       'asynchronous'
%
% OUTPUTS:
% EVFO: number of cells of each possible state for each timestep
% fip: initial position of the fire focus
% finalT: time when the whole forest has been burnt
%
% HIGH-LEVEL ABSTRACT STATES
% E = empty cell without vegetation (0)
% V = vegetation cell (1)
% F = burning cell (2)
% O = burnt cell (3)
%
% TRANSITION RULES:
% Cell with state E will remain as E (empty cell)
% Cell with state V may go to F if there are F neighbors with a
%       probability I or remain V
% Cell with state F may go to O with a probability B or remain F
% Cell with state O will remain as O
%
% LOW-LEVEL STATES FOR IMPLEMENTATION
% Although the high-level conceptual states are E, V, F, and O, we will
% actually implement low-level integer states in the range {0,1,2,3,4}
% The high-level states will then be inferred as follows:
%    Value of 0 is interpreted as empty (E)
%    Value of 1 is interpreted as vegetation(V)
%    Value of 2 is interpreted as burning (F)
%    Value of 3 is interpreted as burnt (O)
%
% INTERACTION TOPOLOGY:
% The grid is rectangular with Moore neighborhoods and two kinds of BC.
%
```

```
% SYNCHRONOUS VERSION:
%    Every timestep, update every cell based on values at previous timestep
%
% ASYNCHRONOUS VERSION:
%    Every timestep, update every cell randomly based on most current values
%

% STUB MADE BY Maggie Eppstein for the SIR model, 2/24/17
% MODIFIED BY Jack Houk and Javier Lobato for the SIR model, 03/03/2018
% Fire Spread implementation by Javier Lobato, 03/26/2018

% Define state variables to make the code more readable
E = 0;
V = 1;
F = 2;
O = 3;

%Map initialization with a full empty map, adding then random vegetation
map = zeros(dim);
for i=1:dim^2
    if rand() <= D % If the probability is less than D, transform the
                   % empty from matrix preallocation into vegetation
        map(i) = V;
    end
end



% FIRE INITIALIZATION Initialize the map as stated by the input
if isnumeric(initializationConf) % If a number is specified
    fip = zeros([initializationConf, 2]);
    % Create a random ordered matrix
    orderMatrix = reshape(randperm(floor(dim)^2), [floor(dim),floor(dim)]);
    % And spark the first 'initializationConf' values with an F
    for k=1:initializationConf
        [i,j] = find(orderMatrix == k);
        fip(k, :) = [i,j];
        map(i,j) = F;
    end
elseif strcmp(initializationConf, 'VFirewall') % Vertical firewall
    map(:,floor(rand*dim)) = E;
    map(floor(rand*dim),floor(rand*dim)) = F;
elseif strcmp(initializationConf, 'HFirewall') % Horizontal firewall
    map(floor(rand*dim),:) = E;
    map(floor(rand*dim),floor(rand*dim)) = F;
end

% Preallocation of the output matrix
EVFO = zeros([4, maxt+1]);

% Store the number of susceptible, infected and recovered in the first step
EVFO(1, 1) = sum(sum(map == 0));
EVFO(2, 1) = sum(sum(map == 1));
EVFO(3, 1) = sum(sum(map == 2));
EVFO(4, 1) = sum(sum(map == 3));

% PLOT INITIAL MAP (see function below)
```

```
[fighandle,plothandle] = plotMapInNewFigure(map);

% Synchronous updating case
if strcmp(sync_async, 'sync')
    % Preallocation of a whole new map to be filled
    newMap = zeros(dim);
    % Loop every iteration time step
    t = 1;
    while sum(sum(map == 2)) ~= 0 %Until the fire is extinguished
        % Loop over the whole map in X and Y
        for x = 1:dim
            for y = 1:dim
                % Get the eight neighbours of the current cell (x,y)
                % The cardinal coordinates will be used: north, south,
                % east, west - plus their respective combinations
                nn = map(x, abs(mod(y-2, dim))+1);
                ss = map(x, abs(mod(y, dim))+1);
                ww = map(abs(mod(x-2, dim))+1,y);
                ee = map(abs(mod(x, dim))+1,y);
                nw = map(abs(mod(x-2, dim))+1, abs(mod(y-2, dim))+1);
                ne = map(abs(mod(x, dim))+1, abs(mod(y-2, dim))+1);
                sw = map(abs(mod(x-2, dim))+1, abs(mod(y, dim))+1);
                se = map(abs(mod(x, dim))+1, abs(mod(y, dim))+1);

                % Absorbing boundary conditions will assume that in the
                % cells that go outside the grid, the value E will be used
                % x-component (the map is left-right in the x-axis)
                if strcmp(BC, 'absorbing')
                    if x == 1
                        nw = E; ww = E; sw = E;
                    elseif x == dim
                        ne = E; ee = E; se = E;
                    end
                    % y-component
                    if y == 1
                        nw = E; nn = E; ne = E;
                    elseif y == dim
                        sw = E; ss = E; se = E;
                    end
                end
                % Store the neighbors in a vector
                neighbors = [nn, ne, ee, se, ss, sw, ww, nw];

                % With current cell and neighbor value (apart from other
                % parameters such as the probability of infection, the
                % duration of infection, the duration of recovery, and the
                % immigration rate) the value of the new cell will be
                % computed and stored in the same position in the newMap
                newMap(x,y) = cellUpdate(map(x,y), neighbors, I, B);
            end
        end

        % Reassign the newMap to the old variable map
        map = newMap;
        % Store the number of susceptible, infected and recovered
        EVFO(1, t+1) = sum(sum(map == 0));
        EVFO(2, t+1) = sum(sum(map == 1));
```

```
        EVFO(3, t+1) = sum(sum(map == 2));
        EVFO(4, t+1) = sum(sum(map == 3));

        set(plothandle,'cdata',map);
        pause(0.01) % Pause to see the map

       % Bail out if the user closed the figure (if only the last iteration
       % is wanted, fighandle will not exist
            if ~ishandle(fighandle)
                % Plot the final map and exit the loop
                plotMapInNewFigure(map);
                title('Final configuration','FontSize', 24)
                break
            end
    t = t + 1; % Add to the next iteration
    end
    EVFO(1, t+1:end) = EVFO(1, t);
    EVFO(2, t+1:end) = EVFO(2, t);
    EVFO(3, t+1:end) = EVFO(3, t);
    EVFO(4, t+1:end) = EVFO(4, t);
    finalT = t+1;

% Asynchronous updating case
else
    t = 1;
    % Loop every iteration time step
    while sum(sum(map == 2)) ~= 0
        % A random order or substitution will be determined with randperm
        % that gives a random set of number, using the one-entry mode for a
        % 2D array
        for i = randperm(dim^2)
            % Converting the one-entry mode for an array into the classical
            % x and y values for a 2D array
            x = 1 + floor((i-1)/dim);
            y = mod((i-1), dim) + 1;
            nn = map(x, abs(mod(y-2, dim))+1);
            ss = map(x, abs(mod(y, dim))+1);
            ww = map(abs(mod(x-2, dim))+1,y);
            ee = map(abs(mod(x, dim))+1,y);
            nw = map(abs(mod(x-2, dim))+1, abs(mod(y-2, dim))+1);
            ne = map(abs(mod(x, dim))+1, abs(mod(y-2, dim))+1);
            sw = map(abs(mod(x-2, dim))+1, abs(mod(y, dim))+1);
            se = map(abs(mod(x, dim))+1, abs(mod(y, dim))+1);

            % Absorbing boundary conditions will assume that in the
            % cells that go outside the grid, the value E will be used
            % x-component (the map is left-right in the x-axis)
            if strcmp(BC, 'absorbing')
                if x == 1
                    nw = E; ww = E; sw = E;
                elseif x == dim
                    ne = E; ee = E; se = E;
                end
                % y-component
                if y == 1
                    nw = E; nn = E; ne = E;
                elseif y == dim
```

```
                    sw = E; ss = E; se = E;
                end
            end
            % Store the neighbors in a vector
            neighbors = [nn, ne, ee, se, ss, sw, ww, nw];

            % With current cell and neighbor value (apart from other
            % parameters such as the probability of infection, the
            % duration of infection, the duration of recovery, and the
            % immigration rate) the value of the new cell will be
            % computed and stored in the same position in the newMap
            map(x,y) = cellUpdate(map(x,y), neighbors, I, B);
        end
        % Store the number of susceptible, infected and recovered
        EVFO(1, t+1) = sum(sum(map == 0));
        EVFO(2, t+1) = sum(sum(map == 1));
        EVFO(3, t+1) = sum(sum(map == 2));
        EVFO(4, t+1) = sum(sum(map == 3));

        set(plothandle,'cdata',map);
        pause(0.01) % Pause to see the map

       % Bail out if the user closed the figure (if only the last iteration
       % is wanted, fighandle will not exist
            if ~ishandle(fighandle)
                % Plot the final map and exit the loop
                plotMapInNewFigure(map);
                title('Final configuration','FontSize', 24)
                break
            end
    t = t + 1;
    end
    EVFO(1, t+1:end) = EVFO(1, t);
    EVFO(2, t+1:end) = EVFO(2, t);
    EVFO(3, t+1:end) = EVFO(3, t);
    EVFO(4, t+1:end) = EVFO(4, t);
    finalT = t+1;
end
end

% Updating of the center cell with the neighbors
 function newState = cellUpdate(center, neighbors, I, B)
    % If the center cell is vegetation
    if center == 1
        burningNeighbors = 0;
        % Counting the fire neighbors
        for neighbor = neighbors
            if neighbor == 2
                burningNeighbors = burningNeighbors + 1;
            end
        end
        % Apply the correspondant probability
        if rand < 1-(1-I)^burningNeighbors
            newState = 2; % Vegetation -> Fire
        else
            newState = 1; % Vegetation -> Vegetation
        end
```

```
    % If the center cell is fire
    elseif center == 2
        if rand < B
            newState = 3; % Fire -> Burnt
        else
            newState = 2; % Fire -> Fire
        end
    % Otherwise, the cell is empty or already burnt
    else
        newState = center;
    end
 end

% Function to plot the map using imagesc()
function [fighandle,plothandle] = plotMapInNewFigure(map)
    fighandle = figure;
    % Specify location of figure
    set(fighandle,'position',[42   256   560   420]);
    % Doesn't truncate a row and column like pcolor does
    plothandle = imagesc(map);
    % Custom colormap definition to show a more real behavior
    customCM = [0.00 0.00 0.00
                0.00 0.98 0.17
                0.95 0.14 0.13
                0.40 0.40 0.40];
    colormap(customCM);
    % Make sure the color limits don't change dynamically
    set(gca,'clim',[0 3]);
    ch = colorbar;
    set(ch,'Ytick',[0 1 2 3],'Yticklabel',{'Empty', 'Vegetation', 'Fire', 'Burnt'})
    % Make sure aspect ratio is equal
    axis('square')
end
```

dataAnalysis.m file

```
% Let's clean the environment and define the variables
clear all, close all, clc
runs = 5;
focus = 20;
dim = 201;
time = 101;

%%
% Calling to the program and evaluation of the system
EVFO = zeros([focus, runs, 4,time]);
locTime = zeros([focus, runs, focus, 3]);

for i=1:focus
    for j=1:runs
        [EVFO(i,j,:,:), locTime(i,j,1:i,1:2), locTime(i,j,1,3)] = fireSpread(dim, 0.8, 0.6, 0.7,
         ↪  time-1, i, 'toroidal', 'async');
    end
```

```
    close all; % Close the figures from time to time
end
close all;

%%
% Statistical computations for state cell fractions
statisticalFractions = zeros([focus,2,4,time]); %Representing: number of focus, mean-std, states, time

for i=1:focus
    for j=1:4
        for k=1:time
            statisticalFractions(i, 1, j, k) = mean(EVFO(i, :, j, k));
            statisticalFractions(i, 2, j, k) = std(EVFO(i, :, j, k));
        end
    end
end

% Statistical computations for times
statisticalTimes = zeros([focus, 2]); %number of focus, mean-std

for i=1:focus
    statisticalTimes(i,1) = mean(locTime(i,:,1,3));
    statisticalTimes(i,2) = std(locTime(i,:,1,3));
end

%%
% Plotting of the evolution in time of the percentage of state-cells
figNo = 1;
x = linspace(0,time, time);

for i=1:focus
    for j=1:runs
        figure(figNo)
        hold on
            plot(x, reshape(EVFO(i,j,1,:)./dim^2, [time, 1]), 'ks-','MarkerFaceColor','k','Markersize',5)
            plot(x, reshape(EVFO(i,j,2,:)./dim^2, [time, 1]), 'gv-','MarkerFaceColor','g','Markersize',5)
            plot(x, reshape(EVFO(i,j,3,:)./dim^2, [time, 1]), 'ro-','MarkerFaceColor','r','Markersize',5)
            plot(x, reshape(EVFO(i,j,4,:)./dim^2, [time, 1]), '^-', 'Color', [0.4, 0.4, 0.4],'MarkerFaceColor',[0.4, 0.4, 0.4],'Markersize',5)
        hold off
    end
    ley = legend('Empty cells','Vegetation cells','Fire cells','Burnt cells','Location','east');
    set(ley,'FontSize',14)
    xlabel('Iteration','FontSize',18)
    ylabel('Percentage of each state','FontSize',18)
    title(['Simulation with ', num2str(i), ' initial fires'],'FontSize',24)
    xlim([0,time-1])
    set(gcf, 'Position', [0 0 1000 600])
    saveas(gcf,['PML',num2str(i),'.png'])
    figNo = figNo+1;
end

%%
```

```
% Plotting of mu-std of the evolution in time of the percentage of state-cells
for i=1:focus
    for j=1:runs
        figure(figNo)
        hold on
            errorbar(x, reshape(statisticalFractions(i, 1, 1, :)./dim^2, [time,1]), reshape(statisticalFractions(i, 2, 1, :)./dim^2, [time,1]), 'ks-','MarkerFaceColor','k','Markersize',5)
            errorbar(x, reshape(statisticalFractions(i, 1, 2, :)./dim^2, [time,1]), reshape(statisticalFractions(i, 2, 2, :)./dim^2, [time,1]), 'gv-','MarkerFaceColor','g','Markersize',5)
            errorbar(x, reshape(statisticalFractions(i, 1, 3, :)./dim^2, [time,1]), reshape(statisticalFractions(i, 2, 3, :)./dim^2, [time,1]), 'ro-','MarkerFaceColor','r','Markersize',5)
            errorbar(x, reshape(statisticalFractions(i, 1, 4, :)./dim^2, [time,1]), reshape(statisticalFractions(i, 2, 4, :)./dim^2, [time,1]), '^-', 'Color', [0.4, 0.4, 0.4],'MarkerFaceColor',[0.4, 0.4, 0.4],'Markersize',5)
            hold off
        xlim([0,time-1])
    end
    ley = legend('Empty cells','Vegetation cells','Fire cells','Burnt cells','Location','east');
    set(ley,'FontSize',14)
    xlabel('Iteration','FontSize',18)
    ylabel('Percentage of each state','FontSize',18)
    title(['Simulation with ', num2str(i), ' initial fires - Mean and standard deviation'],'FontSize',24)
    xlim([0,time-1])
    set(gcf, 'Position', [0 0 1000 600])
    saveas(gcf,['PML_mu_sigma_',num2str(i),'.png'])
    figNo = figNo+1;
end

%%
% Burning time depending on the number of initial fires
focusVec = linspace(1,focus,focus);
figure(figNo)
hold on
for i=1:focus
    for j=1:runs
        errorbar(i,statisticalTimes(i,1),statisticalTimes(i,2),'b','LineWidth',1.2)
    end
end
plot(focusVec, statisticalTimes(:,1),'b','LineWidth',2)
hold off
xlim([0,focus+1])
xlabel('Number of initial fires','FontSize',18)
ylabel('Burning time','FontSize',18)
title(['Burning time depending on the number of initial fires'],'FontSize',24)
set(gcf, 'Position', [0 0 1000 600])
saveas(gcf,['asymptoticBehavior.png'])
figNo = figNo + 1;

%%
% Normalized burning profiles
figure(figNo)
% plotStyle = {'s--','o:','h-','^-.','d:','x--','v-','+-.'}; %report
plotStyle = {'-','-','-','-','-','-','-','-'}; %beamer
```

```
c = flipud(winter(8));

hold on
counter = 1;
for i=[1,2,3,4,5,10,15,20]
    plot(x/find(statisticalFractions(i, 1, 3, :)==0, 1, 'first'), reshape(statisticalFractions(i,
     ↪  1, 3, :)./dim^2, [time,1]),
     ↪  plotStyle{counter},'Color',c(counter,:),'LineWidth',1,'MarkerFaceColor',c(counter,:))
    legendList{counter} = [num2str(i),' initial fires '];
    counter = counter + 1;
end
hold off
ley = legend(legendList);
set(ley,'FontSize',14)
xlim([0,1])
xlabel('Normalized burning time','FontSize',18)
ylabel('Percentage of fire cells','FontSize',18)
title(['Normalized burning profiles'],'FontSize',24)
set(gcf, 'Position', [0 0 1000 600])
saveas(gcf,['burningProf.png'])
```