FireTracking

Jiayuan Chen, Robin Luo

Team 2.7

Github: https://github.gatech.edu/hluo76/CX4230-CSE6730-Project2

Contents

1 Background Problem Statement

On our overheating planet, wildfire is now one of the most terrifying and costly of all natural disasters. It caused a lot of loss and death in the world. Every year, there happens more than 10 times huge wildfires in the world, especially in some places where the plants are wide-covering. In the middle of January, the Austria caused a huge wildfire and a half million hectares of bush and forest was fired which caused 5 billion Austrians dollars loss. It causes a lot of trouble to the Austrian government, such as illness, homeless and so on. As a result, making a model to simulate the wildfire spreading is pretty important for extinguishing a fire. The objective of this project is to present discrete computational models capable of simulating wildfire phenomena. In case of researching the spreading of the wildfire, we are researching the spreading pattern of the wildfire and we think it is pretty similar with the Cellular Automata SIR (CASIR) Model and ordinary differential equation (ODE) based on SIR system to make the simulation of the wildfire spreading. In the SIR system, the S represents the the number of susceptible people, I represent the number of people infected, and R represents the number of people who have recovered. Between each state of SIR, there consists of a system of three coupled non-linear ordinary differential equations which calculate the proportional from the state S to I and state I to R. In this project, we will use the python and Jypter to implement the CASIR and ODE simulation model. For the CASIR model, the model maps the microstructure onto a discrete square lattice, with each square having one of three states: burned, unburned, burning. A transformation rule is then applied to a random cell and based on the state of the cell and its neighbors the switching probability is calculated. For the ODE model, we will use the mathematical knowledge firstly infer the basic expression of the model and use some deep analysis to make the final model expression with the experiments by Jypter. After the simulation, we hope to solve following problem by our project:

1. How to efficiency extinguish the wildfire;
2.How the wildfire would be spreading with the firemen control;
3. Help the government to calculate the potential loss of the wildfire;

2 Cellular Automata SIR model

2.1 Background

In the Cellular Automata SIR (CASIR) Model, a stochastic cellular automata (CA) model is proposed to simulate susceptible infected-removed populations over space and time. Three initial grid configurations are used to compare and contrast the spatiotemporal dynamics of this system: random, center, and patchy.[1] Cellular Automata SIR model is based on the Basis Cellular Automata Model (CA). A cellular automaton is a discrete dynamical system that consists of a regular

network of finite state automata (cells) by applying some deterministic or probabilistic transformation rule,such as Game-of-life by John Conway. The local cells' states change depending on the states of their neighbors. The process is repeated at discrete time steps.[2] In the evey time instance, the cells will automatically update their state with the given rule. It is now used in many discrete simulations with the combination of some other statistical models. The advantages of the cellular automata do not have restrictions in the type of elementary entities or transformation rules employed. Also, they can map such different situations as the distribution of the values of state variables in a simple finite difference simulation. In this project, we are making the CASIR model based on several assumption and statistical formulas with some existing Empirical models.

2.2 Basic Knowledge of CASIR Model

The fire-spread model is based on Rothermel's surface fire-spread model[3]. If the all forest looks as 1 part, then S(t) + I(t) + R(t) = 1. And we defined R0 represents the rate of spread on a flat terrain with no wind. The direction of maximum spread is obtained through a vectorial summation of the dimensionless coefficients for wind speed and slope.[4] The two-dimensional fire-spread rule used is the one described in [5]. Assuming the fire origin to be located at the rear focus of the ellipse, the fire-spread rate can be calculated based on the angle θ , measured from the direction of maximum spread.($\overline{E}$ is the ellipse eccentricity, defined as $1 + 1/LW^2$.

$R(\theta) = R_0 \frac{(1+\overline{E}cos(\theta))}{1+\overline{E}}$

$LW = 0.936e^{50.5U_{eq}/100} + 0.461e^{-30.5U_{eq}/100} - 0.397$

In which U_eq is the wind speed that alone would produce the combined effect of actual wind and terrain slope

2.3 Assumption

[a.] In the fire season, any plants will stop growth, migration and death with other reasons.

[b.] we will calculate the wildfire serious in a specific area by the brightness of the fire

[c.] Assume we use the hour as the time unit and each hour we calculate the serious level of the wildfire of the specific area by calculating the average of the brightness of this area.

[d.]We assume the fire speed per hour is γ, the extinguish speed per hour is λ

[e.]We assume the forest is a square

[f.]Each cell can be in one of several finite states. In this tutorial the states are:

$$X_t(i,j) = \begin{cases} 0 & Unburned \\ 1 & Burning \\ -1 & Burned \end{cases}$$

2.4 Parameters of the system

The parameters that control the behavior of our system are listed below. Some of these parameter describe our system overall, and some are specific to the fire in our system.

W : the width and length of the forest Time : timestamp

duration: time difference.

windspeed : At time T, the wind speed in environment

winddirection: At time T, the wind direction in environment.

livep: The possible of the forest have live plants at begin

empty: The possible of the forest have empty space at begin

fireoff : the possible fire off automatically
centerw: The center location of the forest

2.5 Environment Setup

Please make sure that you need to download all libraries as below. Otherwise, you will run into an error during the simulation.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
import math
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

2.6 CASIR Model Part Tutorial

Every cell of G is a position that is either empty, firing, or unfired.

The cells of the grid are represented with values as below

[2]:
```
Unfired = 0
FIRING = 1
Death = -1
```

Each cell have space of 1*1, and the possible of the live plants at the begining is 0.96 and the empty space at the begining is 0.04. The $P_{plantbegin} + P_{emptybegin} = 1$.

[3]:
```
w = 33
wind_speed = 0
wind_direction = 'e'
Time = 240
center_w = int(w/2 +1)
duration = 1
livep = 1
empty = 1 - livep
fireoff = 0.2
```

To store the grid, we will use a 2-D Numpy array (G) and initialize the grid with 96% plants, 4% empty. We set the Orginal situation as G and make model on the M. The fire will begin at the center of the forest. In the fire simulation, $P_{firing} + P_{unfired} + P_{empty} = 1$.

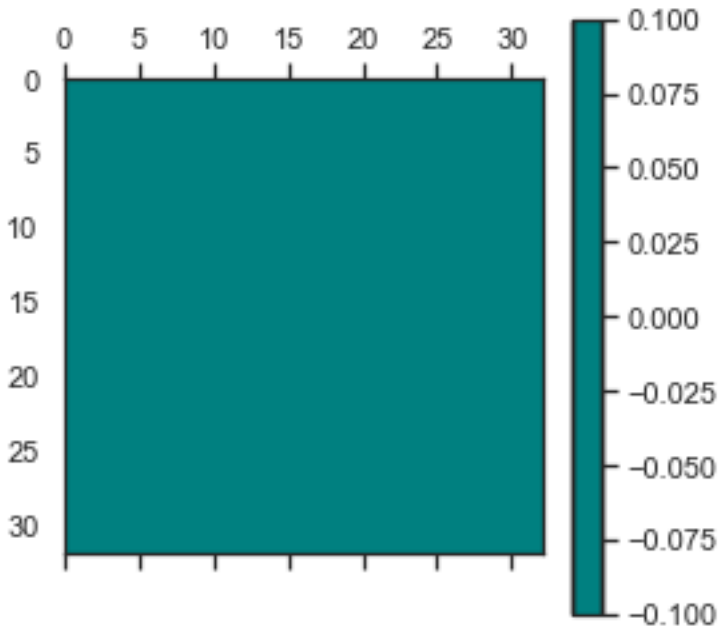
[4]:
```
def create_grid (w, wind_speed, wind_direction, Time, duration, livep,fireoff):
    ### INITIALIZE GRID
    center_w = int(w/2 +1)
    empty = 1- livep
    G = Unfired * np.zeros ((w, w), dtype=int)
    M = Unfired * np.zeros ((w, w), dtype=int)
    #RANDOMLY ASSIGN VALUE 0,1,-1 TO EACH CELL WITH PROB 0.25,0.50,0.25
    for i in range(0,w):
        for j in range(0,w):
```

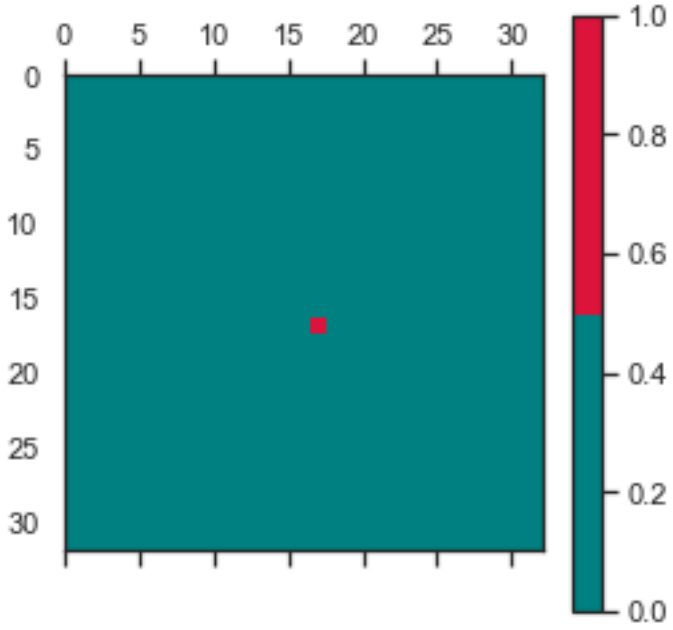
```
            G[i][j]=np.random.choice([0,-1],p=[livep,empty])
            if(G[i][j]!=0):
                M[i][j]= -1
            if (i== center_w and j== center_w):
                M[i][j] = 1
    return G,M, w, wind_speed, wind_direction, Time, duration, livep,␣
 ↪empty,fireoff
```

Draw initial system

[5]:
```
def show_grid ( G,i=1):
    u, count = np.unique(G, return_counts=True)
    if len(count) == 3:
        cmap = mpl.colors.ListedColormap(['skyblue','teal','crimson'])
    elif len(count) == 2:
        cmap = mpl.colors.ListedColormap([ 'teal', 'crimson'])
    else:
        cmap = mpl.colors.ListedColormap(['teal'])
    sns.set(style="white")

    plt.matshow(G,cmap=cmap)
    cb = plt.colorbar ()
    plt.axis ('square')

    plt.axis ([0, w-1, w-1,0])
```

[6]:
```
G1,M1,w, wind_speed, wind_direction, Time, duration, livep, empty,fireoff =␣
 ↪create_grid(w, wind_speed, wind_direction, Time, duration, livep, fireoff)
show_grid(G1)
show_grid(M1)
```





Update the wind influence of the fire spread with the basic knowledge above

[7]:
```
def update_wind(direction):
    LW = 0.936 * np.e ** (50.5* wind_speed/100) + 0.461 * np.e ** (-30.5 *␣
↪wind_speed/100) - 0.397
    E = 1 - 1/LW**2
    theta = 0
    k = 1/(1+E)
    if wind_direction.lower() == 'e':
        return k*(1+E*math.cos(0-direction))
    elif wind_direction.lower() == 'ne':
        return k*(1+E*math.cos(np.pi/4-direction))
    elif wind_direction.lower() == 'n':
        return k*(1+E*math.cos(np.pi/2-direction))
    elif wind_direction.lower() == 'nw':
        return k*(1+E*math.cos(3 * np.pi/4-direction))
    elif wind_direction.lower() == 'w':
        return k*(1+E*math.cos(np.pi-direction))
    elif wind_direction.lower() == 'sw':
        return k*(1+E*math.cos(5 * np.pi/4-direction))
    elif wind_direction.lower() == 's':
        return k*(1+E*math.cos(3 * np.pi/2-direction))
    elif wind_direction.lower() == 'se':
        return k*(1+E*math.cos(7 * np.pi/4-direction))
```

#rule1: if cells contains fire, the rule is as follow:

If >=6 neighbors are firing, fire off If the neighbors is firing, the neighbor fire have 1% possible to off by itself If the neighbors is plants, the plants will be fire with the possible of the 70% multiple the wind influence

[8]:
```
def fire_process(M0, i, j):
    xMin=max(i-1,0)
    xMax=min(i+1,w-1)
    yMin=max(j-1,0)
    yMax=min(j+1,w-1)
    count = 0
    for a in range(xMin,xMax + 1):
        for b in range(yMin, yMax + 1):
            if a == i and  b == j:
                pass
            else:
                if M0[a][b] == -1:
                    pass
                elif M0[a][b] == 0:
                    if a-i == 1 and b == j:
                        direction = np.pi/2
                    elif a - i == -1 and b == j:
                        direction = 3 * np.pi
                    elif a == i and b - j == 1:
                        direction = 0
                    elif a == i and b - j ==-1:
                        direction = np.pi
                    elif a - i == 1 and b - j == 1:
                        direction = np.pi/4
                    elif a - i == -1 and b - j == 1:
                        direction = np.pi/4 * 7
                    elif a - i == 1 and b - j == -1:
                        direction = np.pi/4 * 3
                    elif a - i == -1 and b - j == -1:
                        direction = np.pi/4 * 5
                    possible = 0.7 * update_wind(direction)
                    M0[a][b] = np.random.choice([1,0],p=[possible, 1- possible])
                elif M0[a][b] == 1:
                    count += 1
                    possible_death = 0.5
                    M0[a][b] = np.random.choice([-1,0],p=[possible_death, 1-␣
↪possible_death])
    if count >= 6:
        M0[i][j] = -1
    return M0
```

#rule2: if the cell contains empty, the rule is as follows:

Do nothing

[9]:
```
def empty_process(M0, i, j):
    return M0
```

#rule3: if the cell contains empty, the rule is as follows:
If >= 3 neighbors are firing, plant firing If <3 neighbors are firing, plant have 50% firing

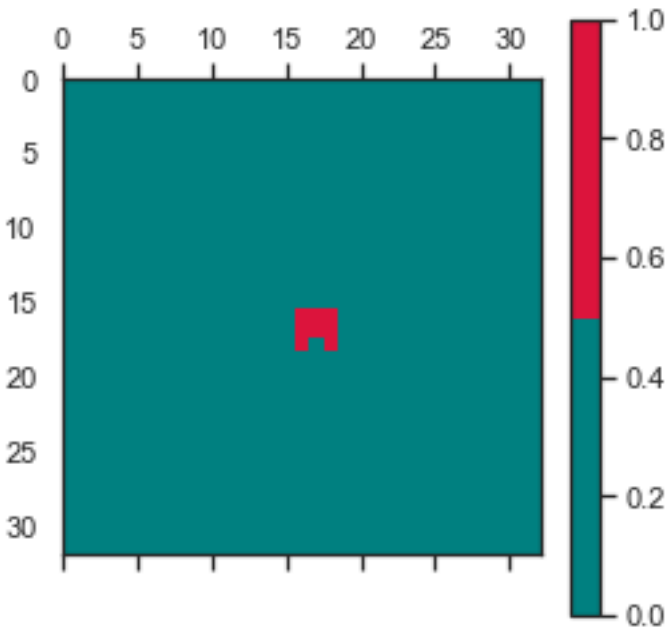
[10]:
```
def plant_process(M0, i, j):
    xMin=max(i-1,0)
    xMax=min(i+1,w-1)
    yMin=max(j-1,0)
    yMax=min(j+1,w-1)
    count = 0
    for a in range(xMin,xMax + 1):
        for b in range(yMin, yMax + 1):
            if a == i and  b == j:
                pass
            else:
                if M0[a][b] == -1:
                    pass
                elif M0[a][b] == 0:
                    pass
                elif M0[a][b] == 1:
                    count += 1
    if count >= 3:
        M0[i][j] = 1
    else:
        np.random.choice([1,0],p=[0.5, 0.5])
    return M0
```

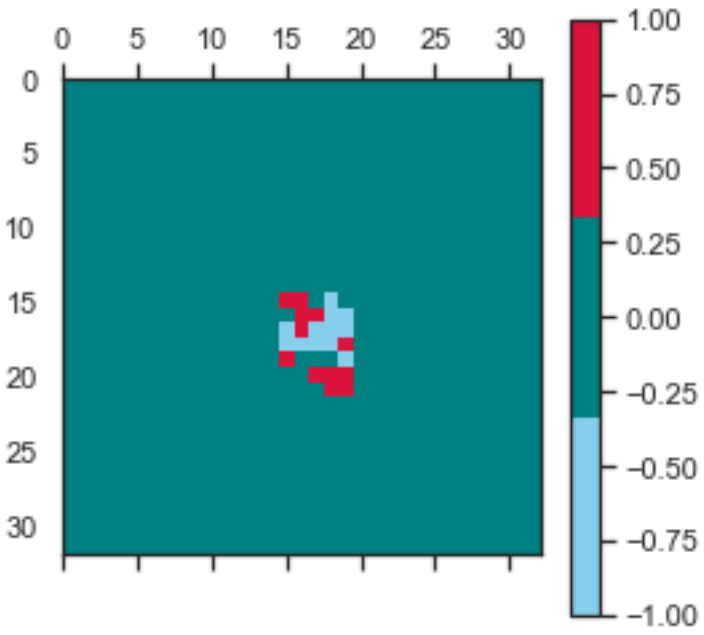
We now shold out the one step of simulation for firing, unfire and empty situation. Iterating through all neighboring cells for each cell takes O(N2) time, since we have N2 cells and check a constant number neighbors. We also have to account for whether there are fixed boundaries (no wraparound) or modular boundaries (grid wraps around horizontally and vertically).

[11]:
```
def step(M):
    fire_point = []
    empty_point = []
    plant_point = []
    M_t = M.copy ()
    for i in range(w):
        for j in range(w):
            if M_t[i][j] == 1:
                fire_point.append((i,j))
            elif M_t[i][j] == 0:
                plant_point.append((i,j))
            elif M_t[i][j] == -1:
                empty_point.append((i,j))
    for k in range(len(empty_point)):
        s = empty_point[k]
```

```
        M_t = empty_process(M_t,s[0], s[1])

    for k in range(len(plant_point)):
        s = plant_point[k]
        M_t = plant_process(M_t,s[0], s[1])

    for k in range(len(fire_point)):
        s = fire_point[k]
        M_t = fire_process(M_t,s[0], s[1])
    return M_t
```

[12]:
```
G_0,M_0, w, wind_speed, wind_direction, Time, duration, livep, empty,fireoff =␣
 ↪create_grid (w, wind_speed, wind_direction, Time, duration, livep, fireoff)
M_1 = step (M_0)
show_grid(M_1)
M_2 = step (M_1)
show_grid(M_2)
```





The preceding code lays the building blocks for the complete simulation, which the following function implements.

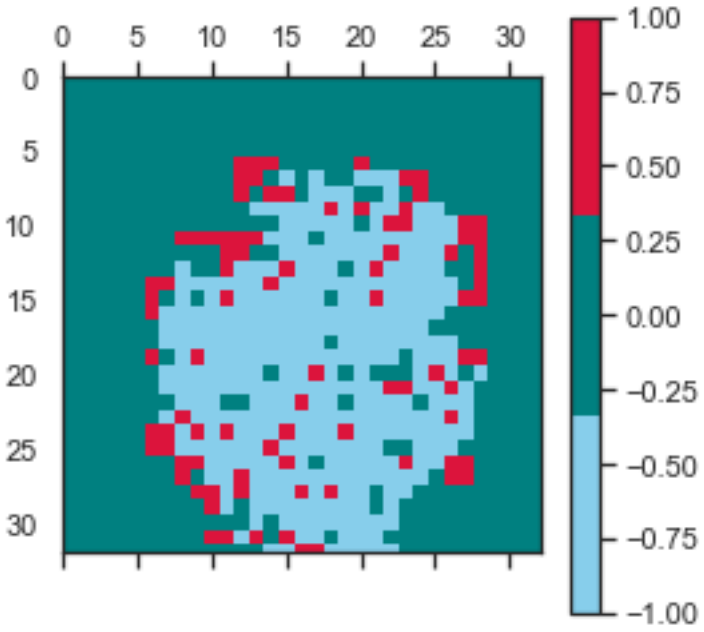
[13]:
```
def sim(M):
    #make the simulation at the system set time
    Nstep = Time
    t, M_t = 0, M.copy ()
    while t < Nstep:
        M_t = step(M)
        t = t + duration
    return (t, M_t)
```

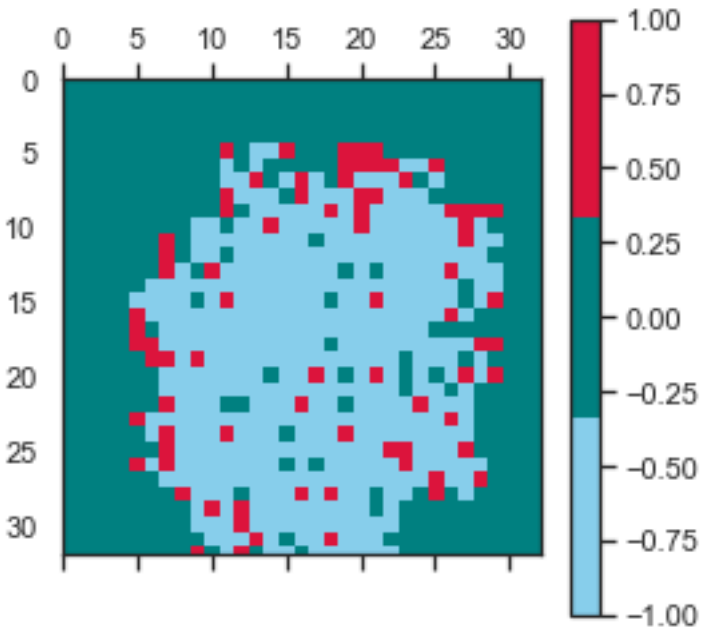
The preceding code lays the building blocks for the specific step simulation, which the following function implements.

[14]:
```
def sim_once(M, Nstep):
    #make more steps of the simualtion
    S = np.zeros (Nstep+1)
    I = np.zeros (Nstep+1)
    R = np.zeros (Nstep+1)
    t, M_t = 0, M.copy ()
    while t < Nstep:
        M_t = step(M_t)
        S[t] = np.count_nonzero(M_t == 0)
        I[t] = np.count_nonzero(M_t == 1)
        R[t] = np.count_nonzero(M_t == -1)
        t = t + duration
    return t,S,I, R, M_t
```

Display the simulation result:

[15]:
```
G1,M1, w, wind_speed, wind_direction, Time, duration, livep, empty,fireoff =␣
 ↪create_grid(w, wind_speed, wind_direction, Time, duration, livep, fireoff)
t,S,I, R, M_t = sim_once(M1, 11)
show_grid(M_t)
t,S,I, R, M_t1 = sim_once(M_t, 1)
show_grid(M_t1)
```





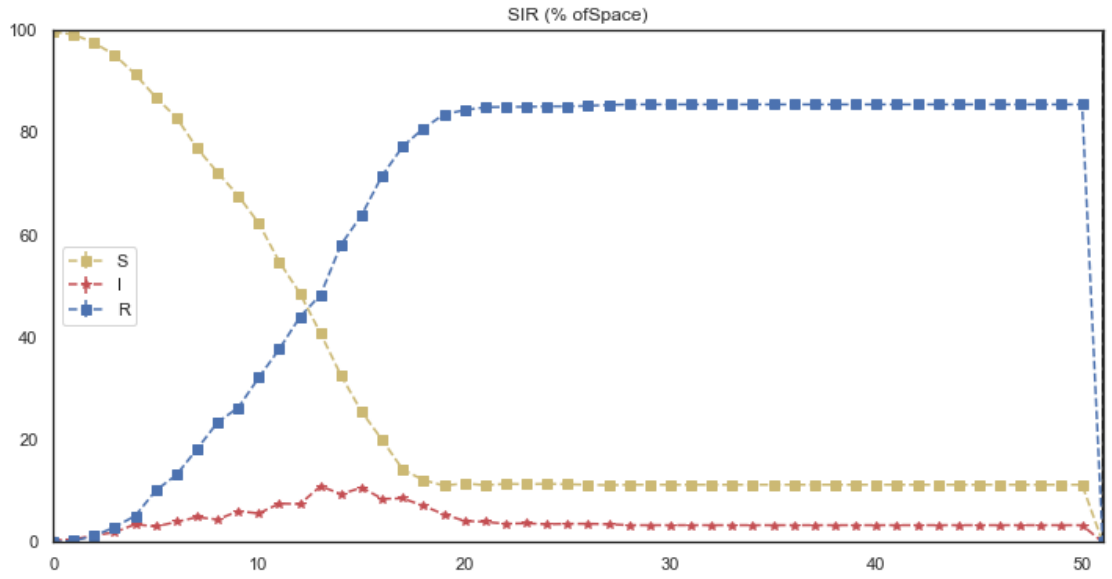
[16]:
```
nIters = 1
I = np.zeros ((52, nIters))
S = np.zeros ((52, nIters))
R = np.zeros ((52, nIters))

T_stop = np.zeros (nIters)
```

[17]:
```
G1,M1,w, wind_speed, wind_direction, Time, duration, livep, empty,fireoff =␣
 ↪create_grid(w, wind_speed, wind_direction, Time, duration, livep, fireoff)
for k in range (nIters): # Loop over simulations
    T_stop[k], S[:, k], I[:, k], R[:,k], M_t = sim_once (M1, 51)
```

[18]:
```
# This code cell helps you visualize your results.

# Computes the averages, $(\bar{S_t}, \bar{I_t}, \bar{R_t})$.
S_avg = np.mean (S, axis=1)
I_avg = np.mean (I, axis=1)
R_avg = np.mean (R, axis=1)
t_stop_avg = np.mean (T_stop)

S_std = np.std (S, axis=1)
I_std = np.std (I, axis=1)
R_std = np.std (R, axis=1)
t_stop_std = np.std (T_stop)

T = np.arange (52)
fig = plt.figure (figsize=(12, 6))
```

```
SCALE = 1e2 / (33**2)
plt.errorbar (T, S_avg*SCALE, yerr=S_std*SCALE, fmt='ys--', label = 'S')
plt.errorbar (T, I_avg*SCALE, yerr=I_std*SCALE, fmt='r*--', label = 'I')
plt.errorbar (T, R_avg*SCALE, yerr=R_std*SCALE, fmt='bs--', label  = 'R')
plt.plot ([t_stop_avg, t_stop_avg], [0., 100.], 'k-')
plt.plot ([t_stop_avg-t_stop_std, t_stop_avg-t_stop_std], [0., 100.], 'k--')
plt.axis ([0, 51, 0.0, 100.0])
plt.legend ()
plt.title ("SIR (% ofSpace)")
```

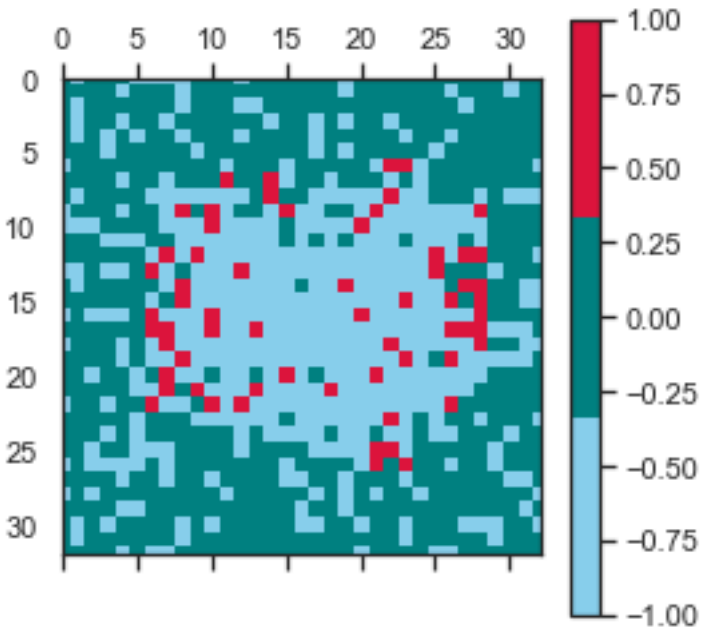
[18]: `Text(0.5,1,'SIR (% ofSpace)')`

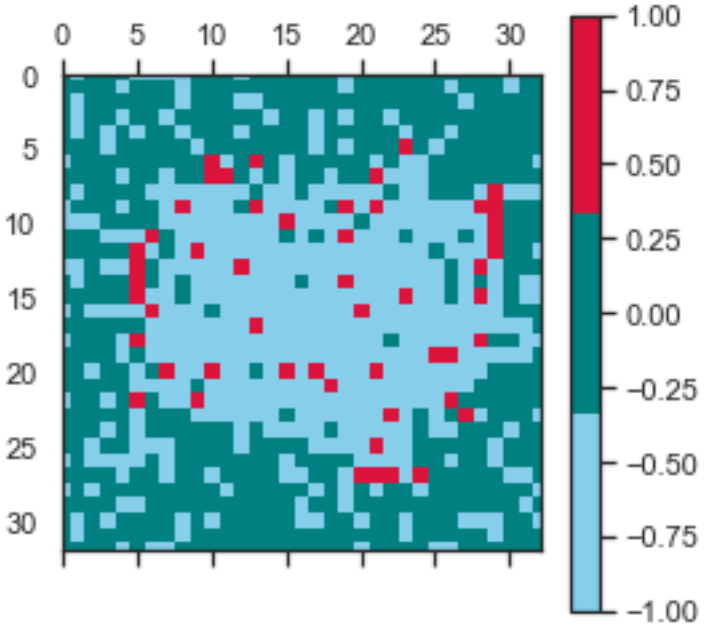


2.7 If there exist some place which is empty at begin

set the emptyp = 0.25

[19]:
```
livep = 0.75
G1,M1, w, wind_speed, wind_direction, Time, duration, livep, empty,fireoff =␣
 ↪create_grid(w, wind_speed, wind_direction, Time, duration, livep, fireoff)
t,S,I, R, M_t = sim_once(M1, 11)
show_grid(M_t)
t,S,I, R, M_t1 = sim_once(M_t, 1)
show_grid(M_t1)
```





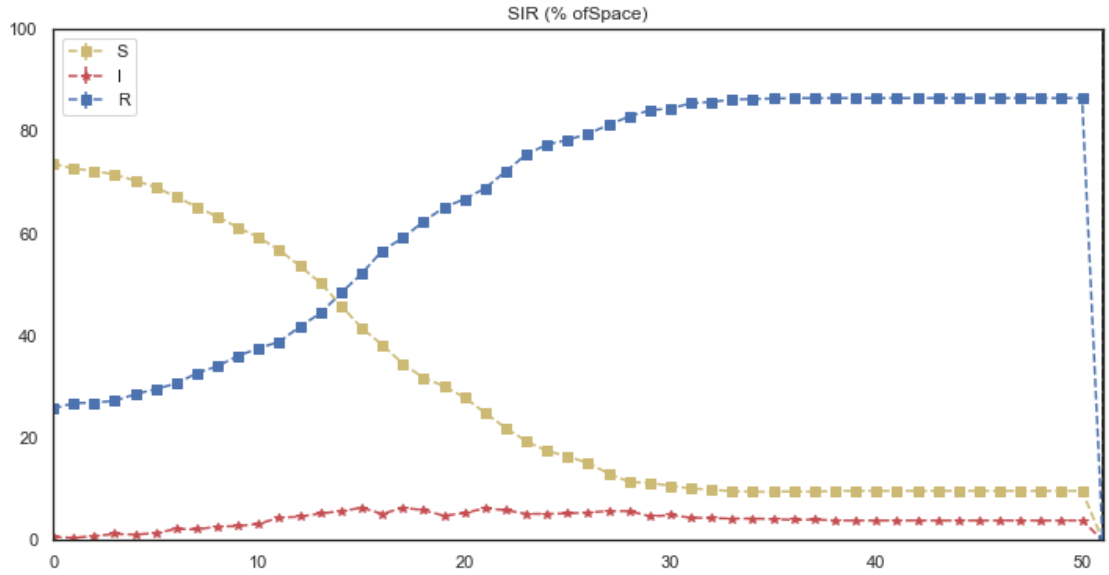
[20]:
```
nIters = 1
I = np.zeros ((52, nIters))
S = np.zeros ((52, nIters))
R = np.zeros ((52, nIters))
```

```
T_stop = np.zeros (nIters)
```

[21]:
```
livep = 0.75
G1,M1, w, wind_speed, wind_direction, Time, duration, livep, empty,fireoff =␣
 ↪create_grid(w, wind_speed, wind_direction, Time, duration, livep, fireoff)
for k in range (nIters): # Loop over simulations
    T_stop[k], S[:, k], I[:, k], R[:,k], M_t = sim_once (M1, 51)
```

[22]:
```
# This code cell helps you visualize your results.

# Computes the averages, $(\bar{S_t}, \bar{I_t}, \bar{R_t})$.
S_avg = np.mean (S, axis=1)
I_avg = np.mean (I, axis=1)
R_avg = np.mean (R, axis=1)
t_stop_avg = np.mean (T_stop)

S_std = np.std (S, axis=1)
I_std = np.std (I, axis=1)
R_std = np.std (R, axis=1)
t_stop_std = np.std (T_stop)

T = np.arange (52)
fig = plt.figure (figsize=(12, 6))
SCALE = 1e2 / (33**2)
plt.errorbar (T, S_avg*SCALE, yerr=S_std*SCALE, fmt='ys--', label = 'S')
plt.errorbar (T, I_avg*SCALE, yerr=I_std*SCALE, fmt='r*--', label = 'I')
plt.errorbar (T, R_avg*SCALE, yerr=R_std*SCALE, fmt='bs--', label  = 'R')
plt.plot ([t_stop_avg, t_stop_avg], [0., 100.], 'k-')
plt.plot ([t_stop_avg-t_stop_std, t_stop_avg-t_stop_std], [0., 100.], 'k--')
plt.axis ([0, 51, 0.0, 100.0])
plt.legend ()
plt.title ("SIR (% ofSpace)")
```

[22]:
```
Text(0.5,1,'SIR (% ofSpace)')
```



2.8 Conclusion

Frome the analysis of the SIRCA Model we create, we observed the pecentage of the plants which not fired will decrease and the firing plants percentage will first increase then decrease, for the plants fired and empty their percentage will increase. It is reasonable because it must have some fire still will not catch off if there are no fireman and rain weather enroll and there are also exist some plants which the surrounding are empty and the fire cannot fire this plant. Also we find with the empty rate at begin increase, the fire rate will decrease and there are more plants will left after the fire. The results are differnt from what we observed in ODEs tutorial where the firing population dynamics exhibited periodic motion. This can be attributed to the limited space and other constraints of the Cellular Automation Model.

3 Ordinary Differential Equations(ODE)

3.1 Background

An ordinary differential equation (ODE) is an equation involving a function (of one variable) and its derivatives. Examples of ODE's are: $y_0 + y = 0$, $\frac{d_x}{d_t} + x^2 t = sin(t)$, and $y'' = xcosy$.

In ODE, many of the expression only can be solved explicitly, only with some special expression can be solved with a actual result. In this SIR system we only can get some explicitly expressions to represent our model.

In our model, for the CASIR model, it is only can represents the roughly situation of the fire spreading. In order to get more realistic relationship of the fire spreading, we use the ODE model to make more mathematical analysis and get some model expressions to show the change thread of the model with those expressions.

3.2 Basic Knowledge of ODE Model

Because, at next timestamp, $t + \Delta t$, the proportion of burning trees is dependent on previous timestamp, t, and, after Δt, the proportion of trees whose status changes from unburned to burning is equal to $f * s(t) * i(t) * \Delta t$ and the proportion of trees whose status changes from burning to burned is equal to $q * i(t) * \Delta t$, then we can get the following equation:

$i(t + \Delta t) = i(t) + f * s(t) * i(t) * \Delta t - q * i(t) * \Delta t$

3.3 Assumption

1. The area under investigation during the period of forest fires does not include population dynamics such as tree growth, death, and migration. The total number of trees, N, is constant.

2. Assume that time is measured in hours.

3. Assume the initial states of $s(0)$, $i(0)$, and $r(0)$ are respectively equal to s_0 which is larger than 0, i_0 which is larger than 0, r_0 should be 0.

3.4 Variables

N: total number of trees including unburned, burning, and burned trees.

t : timestamp

Δ t : time difference.

$s(t)$: at timestamp, t, the proportion of trees that were not burned but could be touched by the fire.

$i(t)$: at timestamp, t, the proportion of burning trees.

$r(t)$: at timestamp, t, the proportion of burned trees whose fire were quenched.

f : the hourly fire rate of trees.

q : the hourly quench rate.

λ : the ratio of f and q, $\frac{f}{q}$.

3.5 Formula

Based on the first assumptions and variables, we can get the following equation.

$s(t) + i(t) + r(t) = 1$

Because, at next timestamp, t + Δ t, the proportion of burning trees is dependent on previous timestamp, t, and, after Δt, the proportion of trees whose status changes from unburned to burning is equal to $f * s(t) * i(t) * \Delta t$ and the proportion of trees whose status changs from burning to burned is equal to $q * i(t) * \Delta t$, then we can get the following equation:

i(t + Δ t) = i(t) + f * s(t) * i(t) * Δ t - q * i(t) * Δ t

Because, at next timestamp, $t+\Delta t$, the proportion of unburned trees is dependent on previous timestamp, t, and, after Δt, the proportion of trees whose status changes from unburned to burning is equal to $f * s(t) * i(t) * \Delta t$, then we can get the following equation:

$s(t + \Delta t) = s(t) - f * s(t) * i(t) * \Delta t$

Because, at next timestamp, $t + \Delta t$, the proportion of burned trees is dependent on previous timestamp, t, and, after Δt, the proportion of trees whose status changes from burning to burned is equal to $q * i(t) * \Delta t$, then we can get the following equation:

$r(t + \Delta t) = r(t) + q * i(t) * \Delta t$

Therefore, based on the third assumption and the above equations, we can get the following differential equations:

$$\begin{cases} \frac{di}{dt} = f * s * i - q * i, & i(0) = i\_0 \\ \frac{ds}{dt} = -f * s * i, & s(0) = s\_0 \\ \frac{dr}{dt} = q * i & r(0) = r\_0 \end{cases}$$

3.6 Environment Setup

Please make sure that you need to download all libraries as below. Otherwise, you will run into an error during the simulation.

[23]:
```
import matplotlib.pyplot as plt
```

3.7 ODE Tuorial

Part 2: Ordinary Differential Equations

2.1 Characteristics of forest fires The occurrence of forest fires has the characteristics of spreading and infectiousness. In a forested area, if a tree catches fire, it can cause surrounding trees to catch fire, and, finally, affects the whole forest. Its main feature is that the simulation of the interaction between individuals in a fixed area is similar to the characteristics of the traditional SIR model of infectious disease research, so the characteristics of the SIR model of infectious disease can be used for reference. Therefore, the SIR model of infectious diseases can be used to study the occurrence and spread of forest fires.

[24]:
```
import numpy as np
import matplotlib.pyplot as plt

N = 1000 #total number of trees
t = 0 #initial timestamp
del_t = 1 #time period on how frequently differential equations
f = 0.5 #hourly fire rate of trees
q = 0.5 #hourly quench rate of trees
lam = f / q
i_0 = 0.1 #initial proportion of burning trees
s_0 = 1 - i_0 #initial proportion of unburned trees
r_0 = 0 #initial proportion of burned trees
nSteps = 100
```

2.4 Formula Based on the first assumptions and variables, we can get the following equation.
$s(t) + i(t) + r(t) = 1$

Because, at next timestamp, $t + \Delta t$, the proportion of burning trees is dependent on previous timestamp, t, and, after Δt, the proportion of trees whose status changes from unburned to burning is equal to $f * s(t) * i(t) * \Delta t$ and the proportion of trees whose status changs from burning to burned is equal to $q * i(t) * \Delta t$, then we can get the following equation:

$i(t + \Delta t) = i(t) + f * s(t) * i(t) * \Delta t - q * i(t) * \Delta t$

Because, at next timestamp, $t + \Delta t$, the proportion of unburned trees is dependent on previous timestamp, t, and, after Δt, the proportion of trees whose status changes from unburned to burning is equal to $f * s(t) * i(t) * \Delta t$, then we can get the following equation:

$s(t + \Delta t) = s(t) - f * s(t) * i(t) * \Delta t$

Because, at next timestamp, $t + \Delta t$, the proportion of burned trees is dependent on previous timestamp, t, and, after Δt, the proportion of trees whose status changes from burning to burned is equal to $q * i(t) * \Delta t$, then we can get the following equation:

$r(t + \Delta t) = r(t) + q * i(t) * \Delta t$

Therefore, based on the third assumption and the above equations, we can get the following differential equations:

$$\begin{cases} \frac{di}{dt} = f * s * i - q * i, & i(0) = i\_0 \\ \frac{ds}{dt} = -f * s * i, & s(0) = s\_0 \\ \frac{dr}{dt} = q * i & r(0) = r\_0 \end{cases}$$

[25]:
```
def burningTreesChange(f, q, s, i, dt):
    di = (f * s * i - q * i) * dt
    return di

def unburnedTreesChange(f, q, s, i, dt):
    ds = (-f * s * i) * dt
    return ds

def burnedTreesChange(q, i, dt):
    dr = q * i * dt
    return dr
```

3.7.1 Using discrete-time steps, to solve the differential equations

We will be using del_t to identify rate change in each timestep. For each simulation run, we will compute the unburned, burned and burning proportions at time dt.

For examples, $i(0)$, $i(1) = i(0) + di(dt)$, $i(2) = i(1) + di(dt)$ and so on.

[26]:
```
s_arr = np.zeros(nSteps + 1)
i_arr = np.zeros(nSteps + 1)
r_arr = np.zeros(nSteps + 1)
```

Simulation step() - Each simulation run will determine change in unburned, burned and burning trees' proportions based on above described equations and will return the new proportions of unburned, burned and burning trees.

[27]:
```
def step(f, q, s, i, r, dt = del_t):
    di = burningTreesChange(f, q, s, i, dt)
    ds = unburnedTreesChange(f, q, s, i, dt)
    dr = burnedTreesChange(q, i, dt)
    s = s + ds
    i = i + di
    r = r + dr
    return s, i, r
```

In one round of simulation, we go through *nSteps* times and in each round the unburned, burned and burning proportion are updated with respect to del_t period.

[28]:
```
def simulation_one_round(f, q, s_0, i_0,r_0, t_max=nSteps):
    s_arr = np.zeros(nSteps + 1)
    i_arr = np.zeros(nSteps + 1)
    r_arr = np.zeros(nSteps + 1)
    t = 0
    s_arr[0] = s_0
    i_arr[0] = i_0
    r_arr[0] = r_0
    while (i_arr[t] > 0) and (t < t_max):
        s_arr[t + 1], i_arr[t + 1], r_arr[t+1] = step(f, q, s_arr[t], i_arr[t],␣
↪r_arr[t])
        t += 1
    # fill in steady-state values, if any
    if t < nSteps:
        s_arr[t+1:] = s_arr[t]
        i_arr[t+1:] = i_arr[t]
        r_arr[t+1:] = r_arr[t]
    return s_arr, i_arr, r_arr
```

2.5 Experiment

Time-series Plot of unburned, burned and burning trees' proportions Next we will draw time-series-plot of unburned, burned and burning trees' proportions to see how the proportion changes with time.

[29]:
```
def show_timeplot(s_arr, i_arr, r_arr):
    T = np.arange(nSteps + 1)


    # plots proportion of unburned trees with a green line and
    # the proportion of burning trees with red line respect to time
    plt.plot(T, s_arr, 'g-', label='proportion of unburned trees')
    plt.plot(T, i_arr, 'r-', label='proportion of burning trees')
    plt.plot(T, r_arr, 'b-', label='proportion of burned trees')

    plt.xlim(0, nSteps) # set x limits of graph
    plt.ylim(0, 1) # set y limits of graph

    # provide labels for the graph
    plt.xlabel('Period (t)')
    plt.ylabel('Proportion')
    plt.title('Evolution of proportions of unburned, burned and burning trees')
    plt.legend()
```

```
    # displays graph
    plt.show()
```

3.7.2 Phase Plot of Proportions of Unburned Trees and Burning Trees

Next we will draw phase-plot of proportions of unburned trees and burning trees to better understand the dynamics.

[30]:
```
def show_phaseplot(s_arr, i_arr, s_0, i_0):
    plt.plot(s_arr, i_arr, "-")
    plt.xlabel("Proportion of unburned trees")
    plt.ylabel("proportion of burning trees")
    plt.title("Proportion of unburned trees vs proportion of burning trees␣
↪(s_0="+str(s_0)+", i_0="+str(i_0)+")")
```

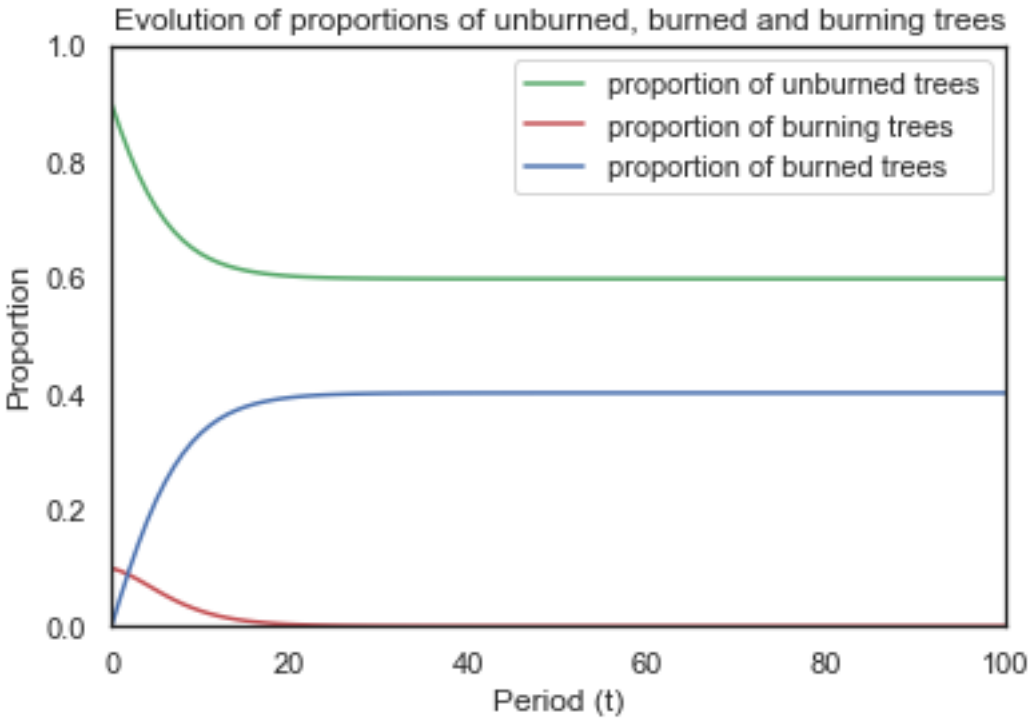
Because $s + i + r = 1$ and $\frac{dr}{dt} = -\frac{ds}{dt} - \frac{di}{dt}$, we can simplify and get the following version:

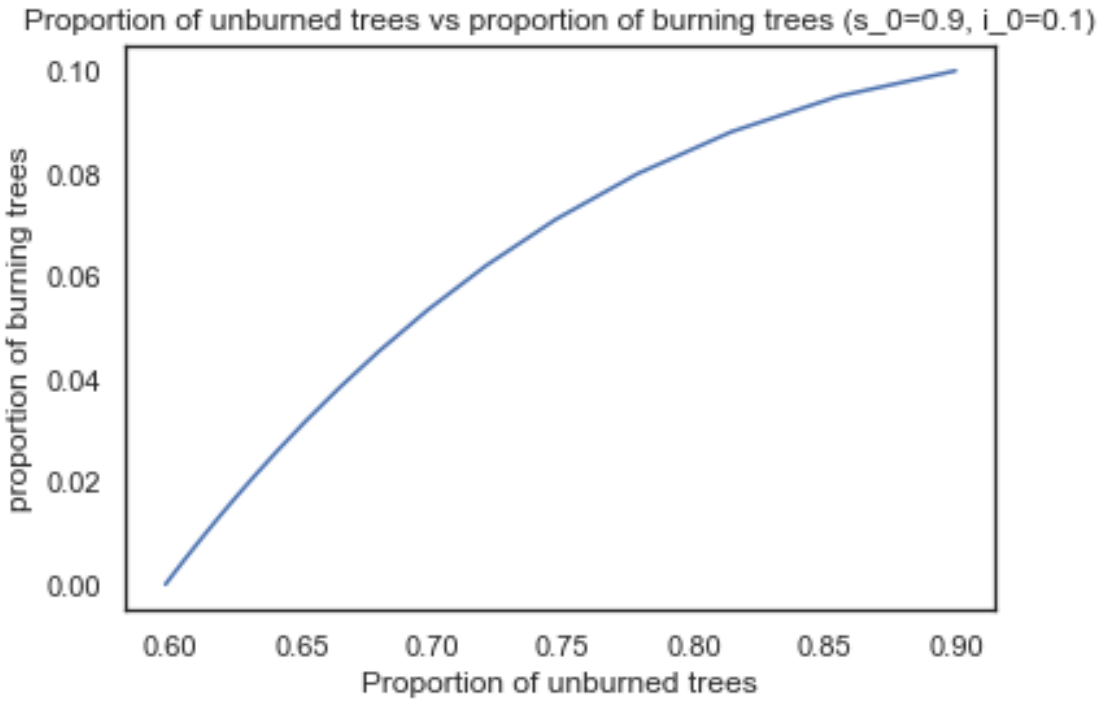
$$\begin{cases} \frac{di}{dt} = f * s * i - q * i, & i(0) = i_0 \\ \frac{ds}{dt} = -f * s * i, & s(0) = s_0 \end{cases}$$

$i_0 + s_0 = 1$, ($r_0 = 0$ based on assumption 3)

Run the simulation with default $s_0 = 0.9$, $i_0 = 0.1$, $f = 0.5$ and $q = 0.5$.

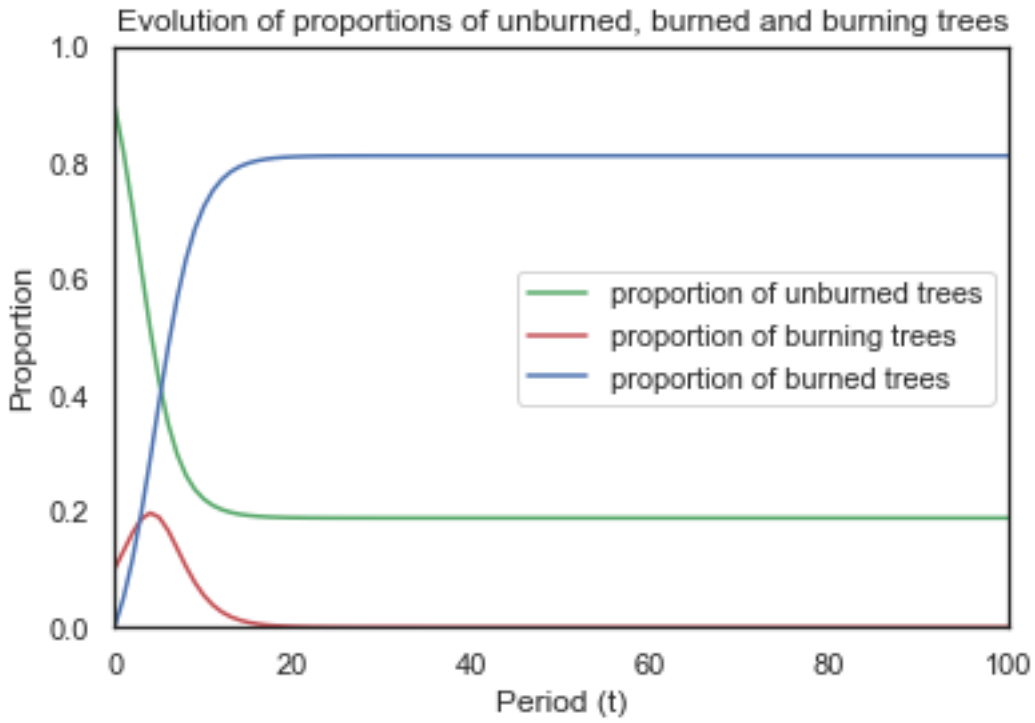
[31]:
```
s_arr, i_arr, r_arr = simulation_one_round(f, q, s_0, i_0, 0)
show_timeplot(s_arr, i_arr, r_arr)
show_phaseplot(s_arr, i_arr, s_0, i_0)
```

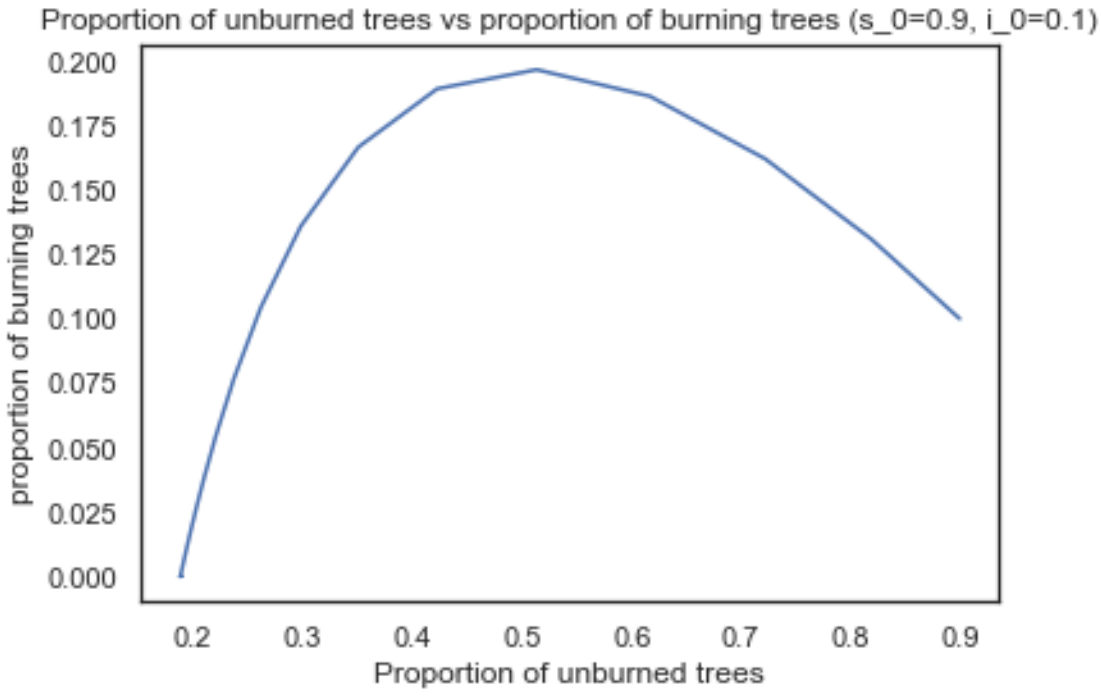




Let's try to increase the f and see what change will happen. Run the simulation with $s_0 = 0.9$, $i_0 = 0.1$, $f = 0.9$, and $q = 0.5$.

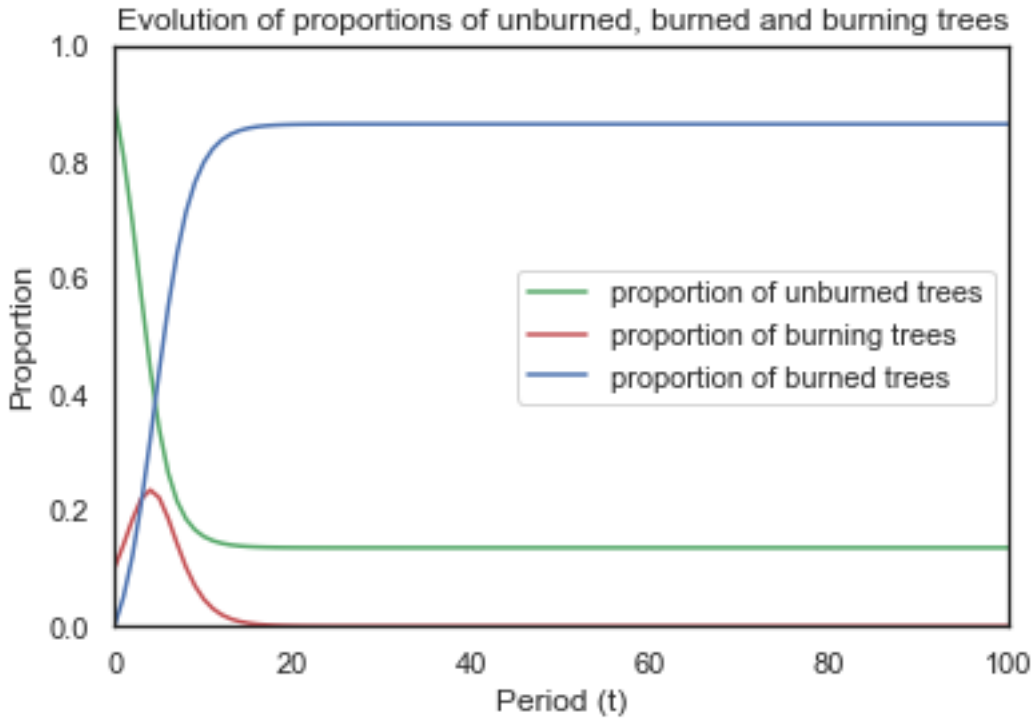
[32]:
```
s_arr, i_arr, r_arr = simulation_one_round(0.9, 0.5, 0.9, 0.1, 0)
show_timeplot(s_arr, i_arr, r_arr)
show_phaseplot(s_arr, i_arr, 0.9, 0.1)
```

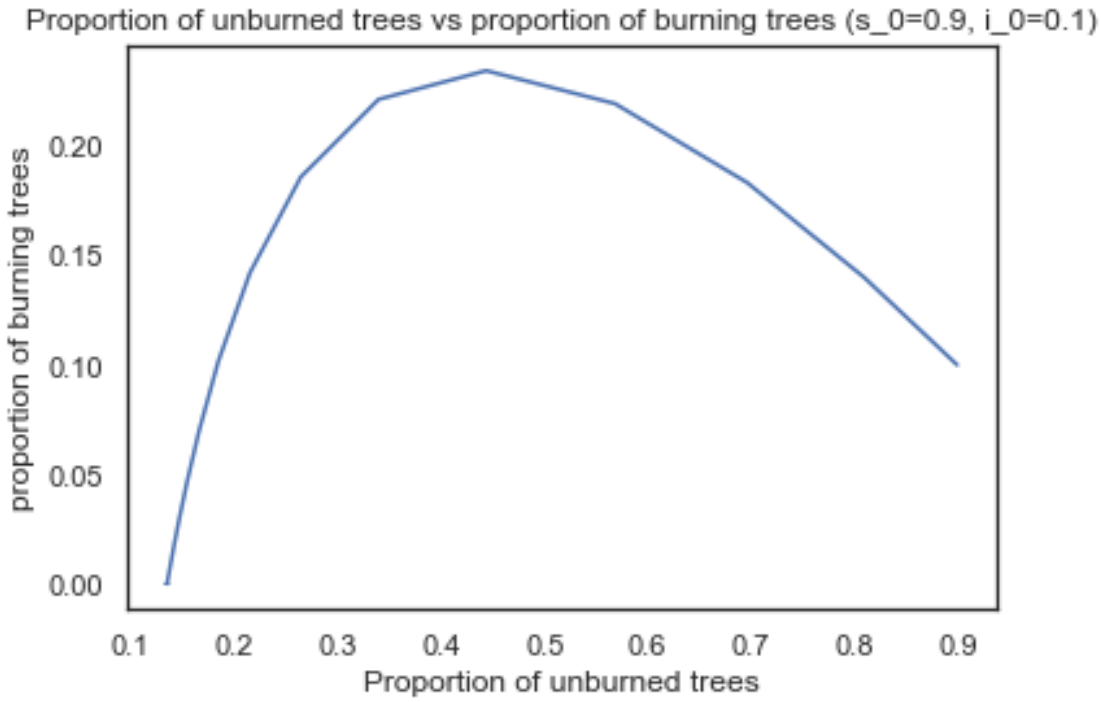




Run the simulation with $s_0 = 0.9$, $i_0 = 0.1$, $f = 1$, and $q = 0.5$.

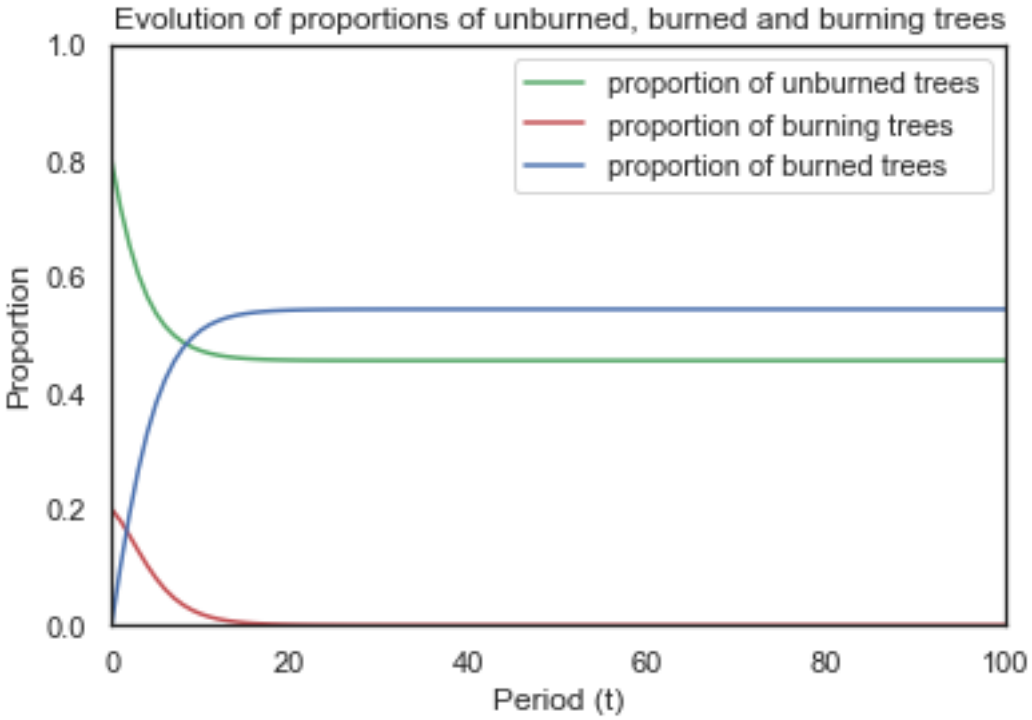
[33]:
```
s_arr, i_arr, r_arr = simulation_one_round(1, 0.5, 0.9, 0.1, 0)
show_timeplot(s_arr, i_arr, r_arr)
show_phaseplot(s_arr, i_arr, 0.9, 0.1)
```

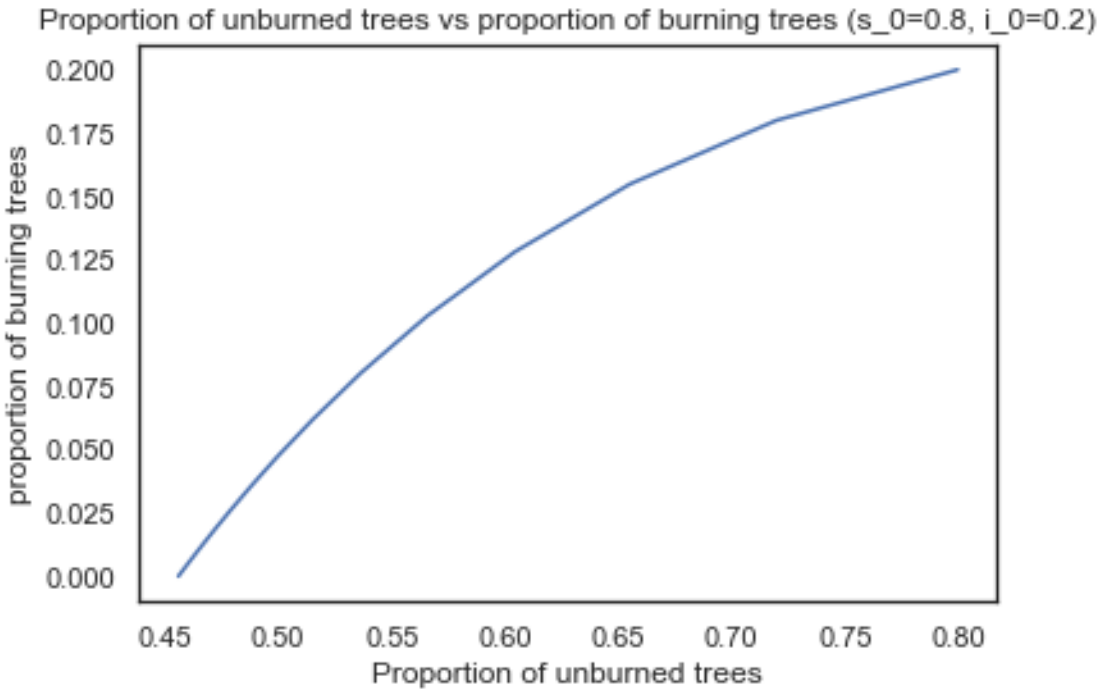




Then try to increase i_0 and then compare with dfault values. Run the simulation with default $s_0 = 0.8$, $i_0 = 0.2$, $f = 0.5$ and $q = 0.5$. ($s_0 + i_0 = 1$)

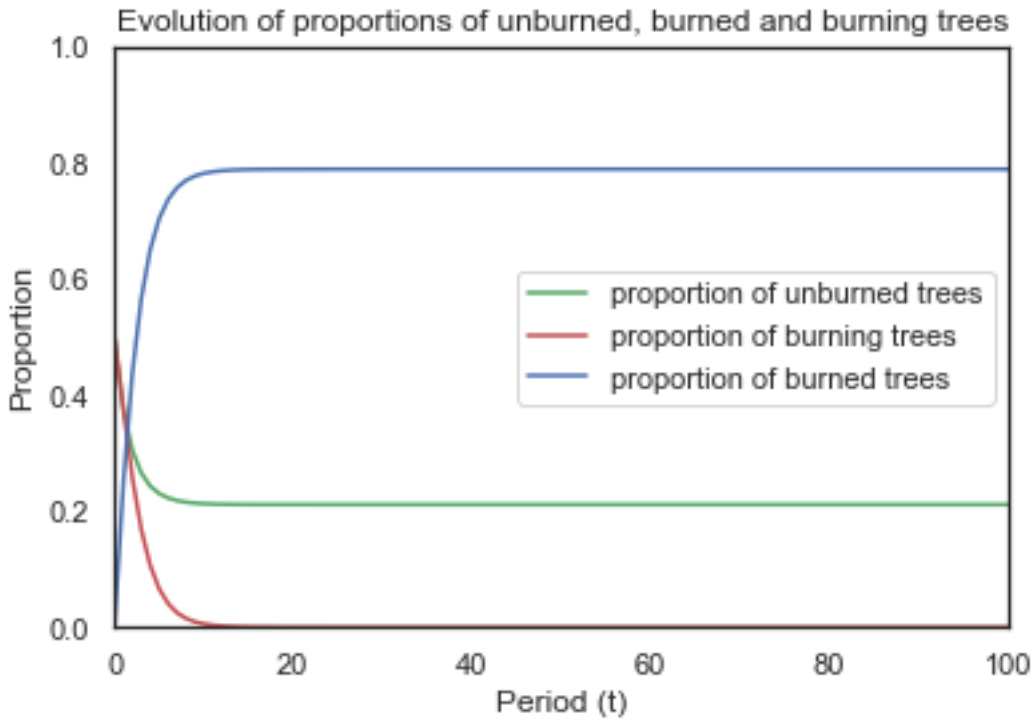
[34]:
```
s_arr, i_arr, r_arr = simulation_one_round(0.5, 0.5, 0.8, 0.2,0)
show_timeplot( s_arr, i_arr, r_arr)
show_phaseplot(s_arr, i_arr, 0.8, 0.2)
```

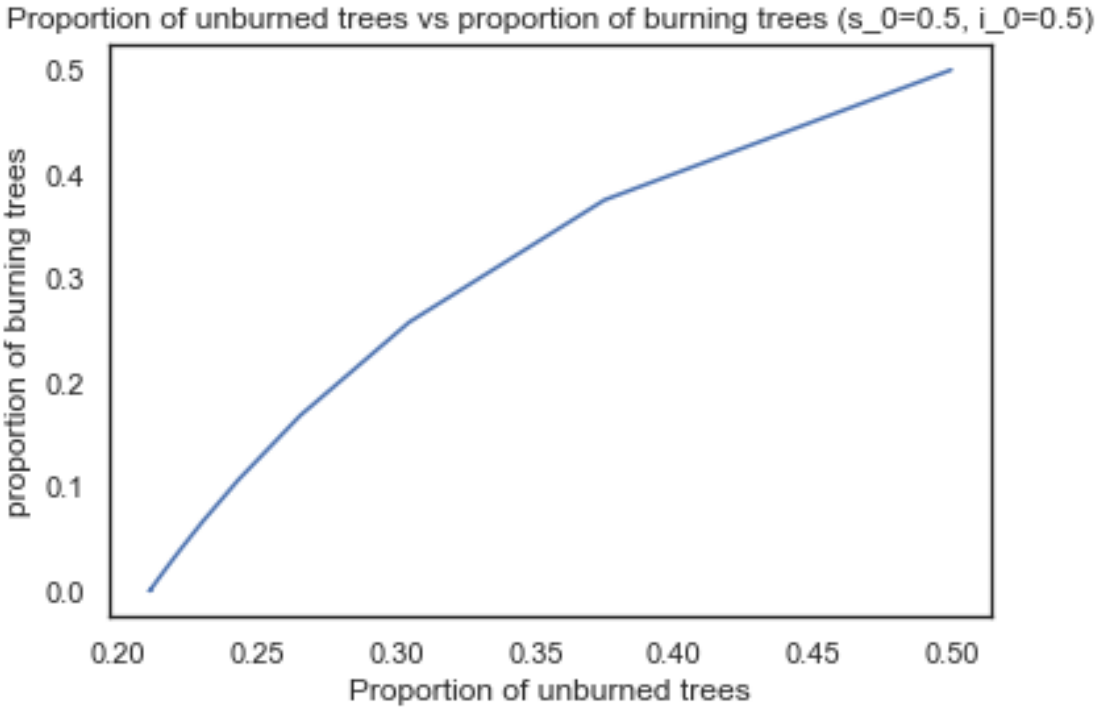




Run the simulation with default $s_0 = 0.5$, $i_0 = 0.5$, $f = 0.5$ and $q = 0.5$. ($s_0 + i_0 = 1$)

[35]:
```
s_arr, i_arr, r_arr = simulation_one_round(0.5, 0.5, 0.5, 0.5, 0)
show_timeplot( s_arr, i_arr, r_arr)
show_phaseplot(s_arr, i_arr, 0.5, 0.5)
```


Evolution of proportions of unburned, burned and burning trees
Proportion
Period (t)
proportion of unburned trees
proportion of burning trees
proportion of burned trees
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100


Proportion of unburned trees vs proportion of burning trees (s_0=0.5, i_0=0.5)
proportion of burning trees
Proportion of unburned trees
0.5
0.4
0.3
0.2
0.1
0.0
0.20
0.25
0.30
0.35
0.40
0.45
0.50

We can found that the relationship between the proportion of unburned trees and the proportion of burning trees looks similar when we just change the s_0 and i_0. But we can see that the relationship between the proportion of unburned trees and the proportion of burning trees changes when we change the f. It seems like the relationship between the proportion of unburned trees and the proportion of burning trees is dependent on f and q. Therefore, for now, we just konw that when increasing the f, the fire will change from not spreading to spreading by looking and comparing the figures of "Proportion of unburned trees vs proportion of burning trees" with different f. When q, s_0 and i_0 are constant, the higher value of f makes the proportion of burning trees increase firstly and then decrese and the rightmost point is the starting point which mean the fire spread firstly and then not spread. But the lower value of f makes the proportion of burning trees always decrese which means fire not spread. However, there are still many possible combinations of f, q, s_0, and i_0 values which we do not experiment. Therefore, we will analyze these four variables more deeply later to figure out their relationship with spread of fire.

We already found that: $\begin{cases} \frac{di}{dt} = f*s*i - q*i,\ i(0) = i\_0 \\ \frac{ds}{dt} = -f*s*i,\ s(0) = s\_0 \\ \frac{dr}{dt} = q*i,\ r(0) = r\_0 \end{cases}$

Combining these two differential equations, we can remove the dt and get the following equation:

$\frac{di}{ds} = \frac{f*s*i-q*i}{-f*s*i}$ $\frac{dr}{ds} = \frac{q*i}{-f*s*i}$

Because $\lambda = \frac{f}{q}$, we can simplify the equation into:

$\begin{cases} \frac{di}{ds} = \frac{1}{\lambda*s} - 1 \\ i|_{s=s_0} = i_0 \\ \frac{di}{ds} = \frac{1}{-\lambda*s} \\ r|_{s=s_0} = r_0 \end{cases}$ Therefore, we can get:

$i(s) = (s_0 + i_0) - s + \frac{1}{\lambda} * ln(\frac{s}{s_0}) = 1 - s + \frac{1}{\lambda} * ln(\frac{s}{s_0})$

($s_0 + i_0 = 1$ based on assuption 3)

$r(s) = r\_0 - \frac{1}{\lambda} * ln(\frac{s}{s_0})$

[36]:
```
def burn_unburn_change(s0, s,lamda):
    i = 1 - s + 1/lamda * np.log(s/s0)
    return i

def burn_burned_change(s0,r0,s, lamda):
    r = r0 - 1/lamda * np.log(s/s0)
    return r
```

In order to research the relationship between the proportions of unburned trees and burning trees with s_0.

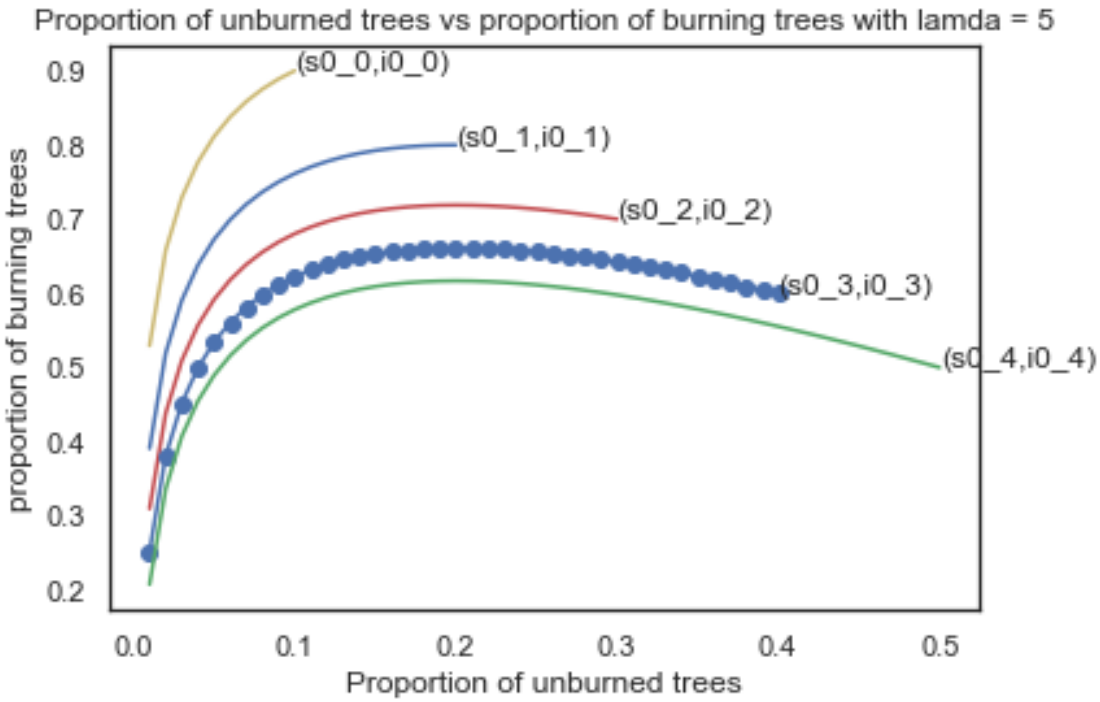
[37]:
```
s0_arr = np.arange(1,6)/10
s_arr = np.arange(100)/100
```

Run the simulation with default ($\lambda = 5$).

[38]:
```
i_arr0 = []
i_arr1 = []
i_arr2 = []
i_arr3 = []
```

```
i_arr4 = []
s_arr0, s_arr1, s_arr2,  s_arr3,  s_arr4 = s_arr.copy(), s_arr.copy(), s_arr.
 ↪copy(), s_arr.copy(), s_arr.copy()
s_arr0 = s_arr0[s_arr0 <= s0_arr[0]]
s_arr1 = s_arr1[s_arr1 <= s0_arr[1]]
s_arr2 = s_arr2[s_arr2 <= s0_arr[2]]
s_arr3 = s_arr3[s_arr3 <= s0_arr[3]]
s_arr4 = s_arr4[s_arr4 <= s0_arr[4]]
for i in range(len(s_arr0)):
    i_arr0.append(burn_unburn_change(s0_arr[0], s_arr0[i], 5))
for i in range(len(s_arr1)):
    i_arr1.append(burn_unburn_change(s0_arr[1], s_arr1[i], 5))
for i in range(len(s_arr2)):
    i_arr2.append(burn_unburn_change(s0_arr[2], s_arr2[i], 5))
for i in range(len(s_arr3)):
    i_arr3.append(burn_unburn_change(s0_arr[3], s_arr[i], 5))
for i in range(len(s_arr4)):
    i_arr4.append(burn_unburn_change(s0_arr[4], s_arr[i], 5))
plt.plot(s_arr0, i_arr0, "y-")
plt.plot(s_arr1, i_arr1, "b-")
plt.plot(s_arr2, i_arr2, "r-")
plt.plot(s_arr3, i_arr3, "o-")
plt.plot(s_arr4, i_arr4, "g-")
plt.text(s_arr0[-1],i_arr0[-1],'(s0_0,i0_0)')
plt.text(s_arr1[-1],i_arr1[-1],'(s0_1,i0_1)')
plt.text(s_arr2[-1],i_arr2[-1],'(s0_2,i0_2)')
plt.text(s_arr3[-1],i_arr3[-1],'(s0_3,i0_3)')
plt.text(s_arr4[-1],i_arr4[-1],'(s0_4,i0_4)')
plt.xlabel("Proportion of unburned trees")
plt.ylabel("proportion of burning trees")
plt.title("Proportion of unburned trees vs proportion of burning trees with␣
 ↪lamda = 5")
```

[38]: `Text(0.5,1,'Proportion of unburned trees vs proportion of burning trees with lamda = 5')`



2.6 Conclusion From our experiment when $s_0 > \frac{1}{\lambda}$. $i(t)$ will first increase then decrease to 0, it means the fire will be spread. If $s_0 \leq \frac{1}{\lambda}$, then it will decrease to 0 directly, which means the fire will not be spread. As a result, we can get the condition of the fire not spread is $s_0 \leq \frac{1}{\lambda}$. After we make the deep analysis of our ODE model, the model start to become similar with our Cellular Automata Model.

4 Division of Labor

4.1 Robin Luo

Focused on implementing the Cellular Automata Model Simulation

4.2 Jiayuan Chen

Focused on implementing the ODE Simulation

5 Acknowledge

We acknowledge the use of data and imagery from LANCE FIRMS operated by NASA's Earth Science Data and Information System (ESDIS) with funding provided by NASA Headquarters.

I. NRT VIIRS 375 m Active Fire product VNP14IMGT. Available on-line [https://earthdata.nasa.gov/firms]. doi: 10.5067/FIRMS/VIIRS/VNP14IMGT.NRT.001.

II. MODISCollection6NRTHotspot/ActiveFireDetectionsMCD14DL.Availableonline[https://earthdata.nasa.gov/fir doi: 10.5067/FIRMS/MODIS/MCD14DL.NRT.006

6 References

[1] Erin L Landguth. A cellular automata sir model for landscape epidemiology. University of Montana, 2007.

[2] Jarkko Kari. Theory of cellular automata: A survey. Theoretical computer science, 334(1-3):3–33, 2005.

[3] Richard C Rothermel.A mathematical model for predicting fire spread in wild land fuels, volume115.Intermountain Forest and Range Experiment Station, Forest Service, United States ,1972.

[4] Mark A Finney. FARSITE, Fire Area Simulator–model development and evaluation. Number 4. US Department of Agriculture, Forest Service, Rocky Mountain Research Station, 1998.

[5] ME Alexander et al. Estimating the length-to-breadth ratio of elliptical forest fire patterns. In Proceedings of the eighth conference on fire and forest meteorology, volume 29, pages 85–04. Soc. Am. For Bethesda, MD, 1985.

[6] Tiziano Ghisu, Bachisio Arca, Grazia Pellizzaro, and Pierpaolo Duce. An improved cellular automata for wildfire spread. Procedia Computer Science, 51:2287–2296,2015.