M20 - MATLAB FINAL PROJECT

Linh Tang

University of California, Los Angeles

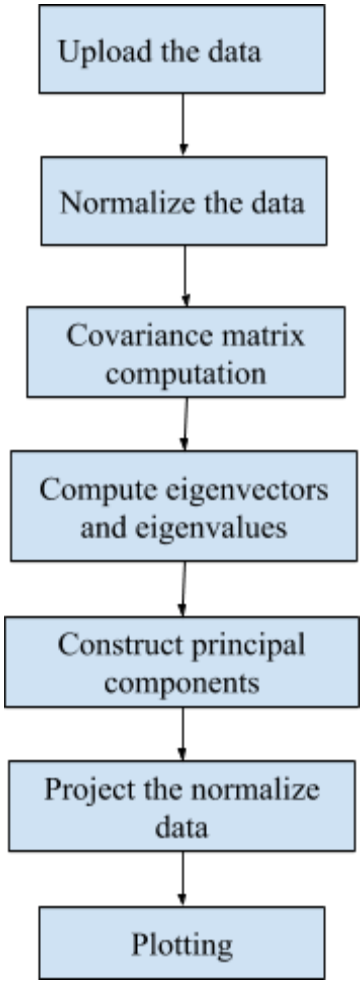
December 11, 2020

1. Introduction

The COVID-19 pandemic has become a great challenge and burden that the world is encountering now; this global pandemic not only has affected public health, but also taken millions lives. In 2020, the pandemic has caused devastating effects on social life and the economy worldwide. Therefore, the study of the spread of disease is essential and useful to predict and visualize the progression of the disease over time. In this project, there are two methods being used to analyze and visualize the COVID- 19 data; they are Principal Component Analysis (PCA) and the Spatial S.I.R model. The goal of this project is to plot the projected data and animate the SIR model by using MATLAB.

2. Principal Component Analysis (PCA)

Principal Component Analysis method is usually used to simplify the complex dimensionality of data sets; this makes the data analysis process easier for machine learning. Even though this method changes a large data set to a smaller data set by changing the basis, it still contains important components and does not lose so much information. Here are steps to build a program which analyze the COVID-19 data and plot the result by using PCA. The program includes a main script and a myPCA function. The main script is for loading the data, calling myPCA function and doing the biplot. myPCA function is used to analyze the data.



Flow charts 1. Steps for PCA program

2.1 Creating myPCA() function

According to the flow charts, this function includes the second step to the sixth step. myPCA function takes the 2D matrix as an input, and returns eigenvectors and the data projected.

```
function [coeffOrth,pcaData] = myPCA(data)
```

2.1.1 Normalizing the data

The goal of this step is to transform the data to another scale where the difference between variables are more comparable; this avoids the biased result [1]. In this case, each column is treated as a variable; values in each column are standardized as following:

$$normalized = \frac{column\ values - mean\ of\ the\ column}{standard\ deviation\ of\ the\ column} \quad (1)$$

In the function, the size of the input data first needs to be obtained by using the built in Matlab function size(); this function returns the number of rows and columns as a row vector.

```
[n, p] = size(data);
```

Before normalizing the data, an array for the normalized data is initialized which has the same size as the input matrix by using zeros() function.

```
normalized = zeros(n, p);
```

After that, a for loop is used to go over columns to get the normalized data for each column. In the calculation, mean() function is used to calculate the mean of the column; std() function is used to calculate the standard deviation of the column; these functions all take each column array as their inputs. The next step is getting the normalized matrix by following the above equation (1).

for j = 1:p (going through p columns)

Calculate the mean of the column

Calculate the standard deviation of the column

Implement equation (1) to calculate the normalized value.

end

2.1.2 Computing covariance matrix, eigenvectors and eigenvalues.

Covariance matrix computation helps us to identify the correlations between variables; how they are varying from the mean [1]. In matlab, there is a built in function called cov() which returns the covariance matrix from the normalized matrix. To get the eigenvectors and eigenvalues of the covariance matrix, eig() function is used.

```
[V, D] = eig(cov(normalized));
```
(V is eigenvectors and D is eigenvalues)

2.1.3 Principal components

Principal components are determined by finding which components have the large possible variance in the data set. Based on linear algebra, the dominant eigenvalues can determine the long term behavior of the system. The greater absolute value of the eigenvalue is, the more importance of that corresponding component. Therefore, the eigenvalues of the matrix needs to be sorted in descending order. Based on that order, eigenvectors are rearranged since eigenvectors decide the direction of the components; what we need is the first two eigenvectors which are corresponding to the most two important components.

Since eigenvalues are compared based on the absolute value, abs() function is called to obtain the absolute value of eigenvalues.

```
D = abs(D);
```

Then, sort() function is used to arrange the eigenvalues in descending order and returns the index of these elements. To make the sorting easier, diag() function is called to convert the main diagonal values in the eigenvalues matrix to a column vector.

```
[~, index] = sort(diag(D),'descend');
```

Lastly, the eigenvectors are arranged according to the return indexes obtained from the sort function.

```
coeffOrth = V(:,index);
```

2.1.4 Determining the project data

The project data is calculated based on the final set of eigenvectors (coeffOrth) and normalized data by using matrix multiplication. A for loop is necessary to calculate

the project data matrix row by row. The size of the project data is the same with the original data.

```
pcaData = zeros(n,p);
for k = 1: n
   pcaData(k,:) = normalized(k,:) * coeffOrth;
end
```

In the end, the myPCA() function returns the eigenvectors (coeffOrth) and the project data (pcaData).

2.2 Main script

2.2.1 Loading the data

The data of 27 countries is given as a .csv file which has 28 rows and 8 columns; however, the program only needs the numerical data from cell C2 to cell H28. To exclude unnecessary cells, setting up the range while loading the data is important in this step. In MATLAB, the function readmatrix() is used to get the data from the excel file. This step is done in the main script.

```
data = readmatrix('covid_countries.csv', 'Range', 'C2:H28');
```

This step returns a 2D matrix which has 6 columns and 27 rows. Six columns are for six variables which are: infections, deaths, cures, mortality, cure rate, infection rate. Twenty seven rows represents the data for twenty seven countries.

2.2.2 Performing PCA method

To make the main script simple, a myPCA function is created separately to do all the algorithms of the method. Calling that function in the main script is the way to obtain the eigenvectors and the project data by just one line of code.

```
[coeff, projectData] = myPCA(data);
```

2.2.3 Biplot

In this part, the problem only requires to plot and visualize the first two components; they are also the first two components in the eigenvectors and project data matrices. This plot shows variables and observations in a 2D vector subspace plot.

```
biplot(coeff(:,1:2),'Score', projectData(:,1:2), 'VarLabels',vbls);
```

2.3 Results

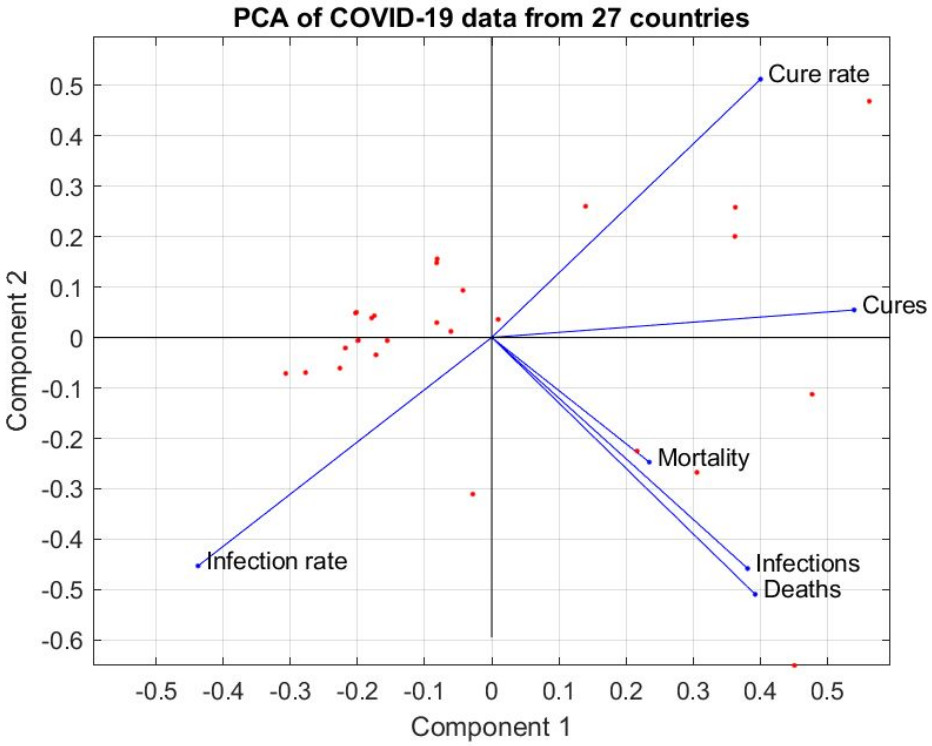


Figure 1. The PCA of Covid-19 data from 27 countries.

2.4 Discussion

According to figure 1, only infection rate has negative coefficient while other five variables have positive coefficient for Component 1. The first principal component

distinguishes observations in the way: if we have high values for infection rate, the rest variables have low values, and vice versa.

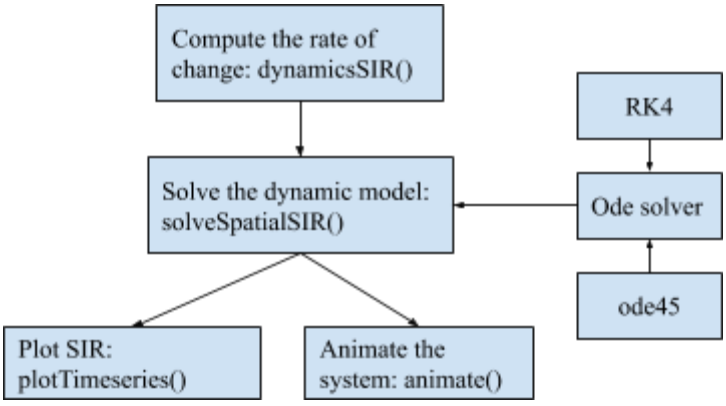
For the second principal component, cure rate and cures have positive coefficient while mortality, infections, deaths, and infection rates have negative coefficient. If we have high values for cure rate, cures variables, then we have low values for mortality, infections, deaths, and infection , and vice versa.

The infection rate is negatively correlated with the cure rate since they form a large angle (about 180°). By forming small angles, the mortality is positively correlated with the infections; the infections is positively correlated with the death.

3. Spatial S.I.R

S.I.R is a common model used to study the spread of diseases which S stands for susceptible individuals, I is for infected individuals and R is for recovered individuals. Spatial S.I.R divides the system into a MxN grid, and each grid has its own local SIR system [2]. By doing this, we can observe the spreading effect over time. In this project, we create our own function to solve differential equations (SIR equations) using Runge Kutta 4 algorithms. In the end, the program displays the SIR plots at a specific location and animates the SIR model. Besides, we also compare the time elapsed between the RK4 solver and the Matlab built in solver ode45.

Here is the flow charts for the Spatial SIR program:



Flow charts 2. Functions in the spatial SIR program

3.1 Runge Kutta - ode solver

Beside Euler methods, Runge Kutta is another method used to solve ordinary differential equations. The fourth order Runge Kutta is the most accurate method, compared to RK1 and RK2. The function takes three inputs: f - a function handle , tspan - a 1x2 time period array which contains start time and final time, y0 - an initial condition.

```
function [t, y] = RK4(f, tspan, y0)
```

According to the instruction file [2], the algorithm of the RK4 is shown as below:

$k_1 = \mathrm{h} * \mathrm{f}(t_n, y_n)$

$k_2 = \mathrm{h} * \mathrm{f}(t_n + \mathrm{h}/2,\ y_n + 1/2 * k_1)$

$k_3 = \mathrm{h} * \mathrm{f}(t_n + \mathrm{h}/2,\ y_n + 1/2 * k_2)$

$k_4 = \mathrm{h} * \mathrm{f}(t_n + \mathrm{h},\ y_n + k_3)$

$t_{n+1} = t_n + \mathrm{h}$

$y_{n+1} = y_n + 1/6\,(k_1 + 2k_2 + 2k_3 + k_4).$

Pseudocode for this function:

Get start time → ti

Get the final time → tf;

Define the step size → h;

Number of steps → nSteps = (tf - ti) / h;

Get the length of initial condition → n

Initialize the solution array → y (size is nSteps x n)

Initialize the time array → t (size is nSteps x 1)

Store the initial condition y0 and ti → yk = y0, tk = ti

Initialize the current step → k = 0

while k <= nSteps

Implement the RK4 algorithm

Update the timestep: k = k + 1

Store tk and yk at the current k into the t and y array

end

3.2 The dynamic model

The dynamicsSIR () function is used to implement SIR dynamic equations shown in the instruction file. Since the both ode solvers RK4 and ode45 only solve the system as they are in a vectorized state (single dimension), the output will also be a vectorized time derivative of state. The function takes 5 inputs: a vectorized state for SIR, size of the grid (M, N), three parameters for the equations.

```
function dxdt = dynamicsSIR(x, M, N, alpha, beta, gamma)
```

SIR dynamic equations [2]

$$\mathrm{d}S_{x,y}(t)/\mathrm{dt} = -(\beta I_{x,y}(t) + \alpha \sum_{i,j} \mathrm{W}(i, j) I_{x+i,y+j}(t)) S_{x,y}(t),$$

$$\mathrm{d}I_{x,y}(t)/\mathrm{dt} = (\beta I_{x,y}(t) + \alpha \sum_{i,j} \mathrm{W}(i, j) I_{x+i,y+j}(t)) S_{x,y}(t) - \gamma I_{x,y}(t),$$

$$dR(t)/dt = \gamma I_{x,y}(t),$$

where α is the spatial contact rate, β is the contact rate, γ is the infectious rate. According to the instruction pdf, W(i,j) is " a weighting function that describes the proximity of nearby neighbors) [2]. Based on the position in the grid, there are three cases which return different numbers of neighbors. The value of W at a specific position is a constant, and the value is defined as:

i,j: row and column in the grid

W(i, j): the weighting at current position.

Case 1: 4 corners (each one has 3 neighbors)

0 W(i,j)	1 W(i, j + 1)	
1 W(i+1, j)	$\frac{1}{\sqrt{2}}$ W(i+1, j + 1)	

Case 2: Boundaries: 1st column, last column, 1st row, last row (except 4 corners)

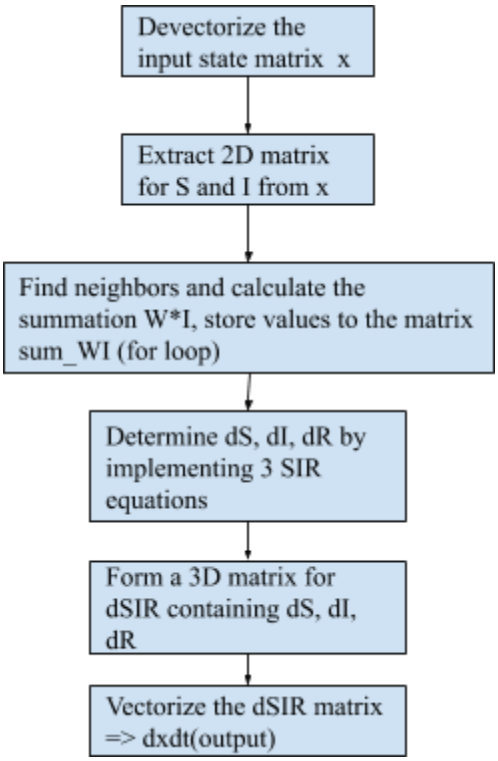
5 neighbors

1 W(i, j-1)	0 W(i,j)	1 W(i, j+1)
$\frac{1}{\sqrt{2}}$ W(i+1, j - 1)	1 W(i+1, j)	$\frac{1}{\sqrt{2}}$ W(i+1, j - 1)

Case 3: Cells at the middle have 8 neighbors.

$\frac{1}{\sqrt{2}}$ W(i-1, j - 1)	1 W(i-1, j)	$\frac{1}{\sqrt{2}}$ W(i-1, j + 1)
1 W(i, j - 1)	0 W(i,j)	1 W(i, j + 1)
$\frac{1}{\sqrt{2}}$ W(i+1, j - 1)	1 W(i+1, j)	$\frac{1}{\sqrt{2}}$ W(i+1, j + 1)

Flow charts for this function:



Flow charts 3. Steps for dynamicsSIR function

For the finding neighbors step, the cell number of neighbors is related to the original cell number: index $\pm M$, index ± 1, index $\pm M \pm 1$ (M is number of rows).

3.3 Solving the SIR dynamic model

The goal of the solveSpatialSIR () function is to solve the dynamic SIR system by calling an ODE solver, either RK4 or ode45. The function takes six inputs: final time, initial condition - a 3D matrix, three SIR parameters (alpha, beta, gamma), an ode solver. The function returns a time array, and the result as a 4D matrix (the state vs time).
Both RK4 function and ode45 take a vectorized state as an input and return the result as a 2D T x n matrix (n is the dimension of y); however, the output of solveSpatialSIR () is a 4D matrix. Therefore, a few steps are required to reorder the dimensions.

Pseudocodes:

Get the size of initial condition matrix → M, N, Z

Vectorize the initial condition matrix

Create 2x1 time period array → tSpan = [0 tFinal]

Create a function handle for the dynamicsSIR → dSIR

Use odesolver to solve dSIR →

[t,x] = odeSolver(dSIRdt,tspan, initialCondition);

Get the size of time array t → T

Devectorize the result x using reshape() to get T x M x N x Z matrix

Reorder the dimensions of x using permute() to get T x M x N x Z matrix

3.4 Time series plotting

The plotTimeSeries function displays the plot of three states S(t), I(t), R(t) at a specific coordinate on the grid. This function takes 4 inputs: a vector of time steps, a 4D matrix representing a local S.I.R model, x - coordinate, y - coordinate. The function has no outputs, only displays a plot for S, I and R.

```
function plotTimeSeries(t, X, x, y)
```

Here are steps for creating this function:

- Get the length of the vector t → timeSteps.
- Initialize 3 arrays with size 1 x timeSteps → S, I, R
- A for loop to get and store values of S(x,y), I(x,y), R(x,y) at those time steps from 4D matrix X. The first two dimensions are the for x,y - coordinates, third dimension is for 3 states S, I, R; fourth dimension specifies the timesteps.

```
for i = 1:timeSteps
        S(i) = X(x,y,1,i);
        I(i) = X(x,y,2,i);
        R(i) = X(x,y,3,i);
  end
```

- Plotting S(t), I(t) and R(t) in the same figure using the subplot() function.

3.5 Animation

The purpose of animate() function is to show the animation showing the spread of disease; blue is for susceptible individuals, red is for infected individuals, green is for recovered individuals [2]. Similar to the plotTimeSeries() function, this function has no output, but this one only takes one input which is the 4D matrix X.

This function requires a for loop to display the image every 10 timesteps; therefore the increment of the time step will be 10. The pause() function is called in the end of this loop to pause 0.1 seconds before changing to the next frame. The size(X,4) returns the length of the fourth dimension in matrix X which is also the number of time steps.

The animation is done by using the image() function which takes a 3D matrix MxNx3. As we know, M and N are the dimensions of the grid; the third dimension

represents the red-green-blue triplet [2]. Before getting into that, the values of S, I and R need to be normalized by dividing each one by the maximum of the state.

Pseudocodes:

for t = 1:10:size(X,4)

Initialize 3D matrix → color

Extract S(t), I(t), R(t) from the 4D matrix - X → St, It, Rt

Normalized St, It, Rt → St = St/max(St(:)); It = It/max(It(:));

Rt = Rt/max(Rt(:)).

Assign red for It, green for Rt, blue for St

→ `color(:,:,1) = It;`

`color(:,:,2) = Rt;`

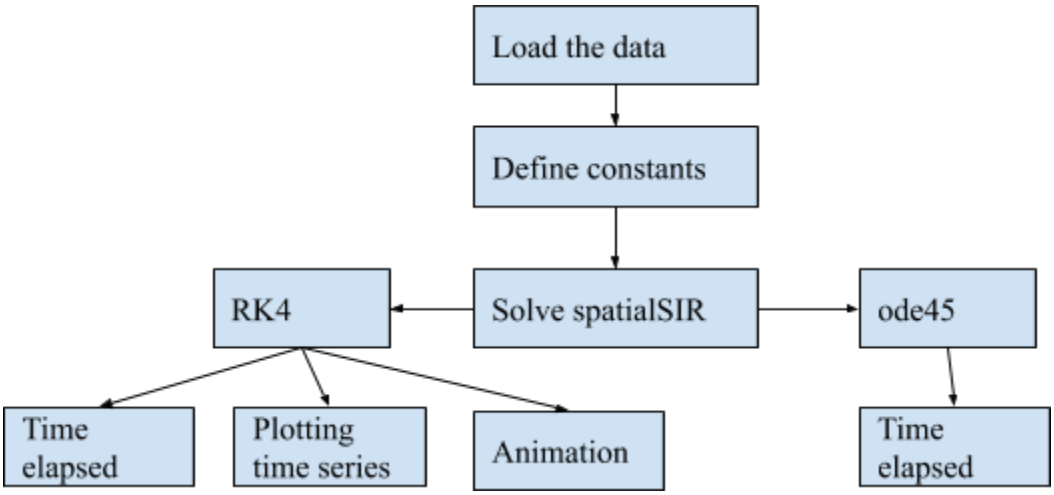
`color(:,:,3) = St;`

Call the image function

Pause for 0.1s

end

3.6 Main script



Flow charts 4. Steps in the main script for spatial SIR program

In the mainscript two times, one is for using ode45 to solve the system, and one is for using RK4. We compare the runtime between them by using tic toc command for each one. For the system solved with RK4, there are two more steps required; plotting and animation.

3.7 Results

- Time elapsed:

  Run time for using ode45 solver: 0.2468s.

  Run time for using RK4 solver: 1.3021s.

- Plots:

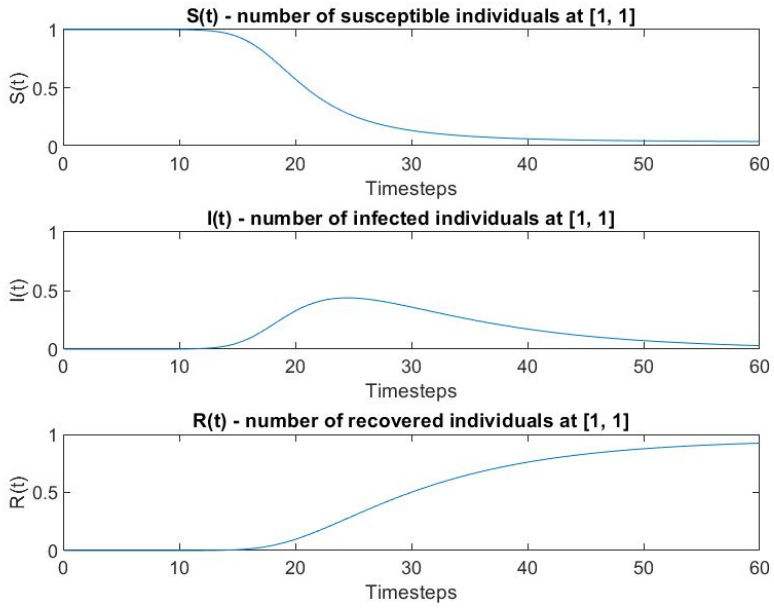


Figure 2. The SIR plots over time at (1,1) position

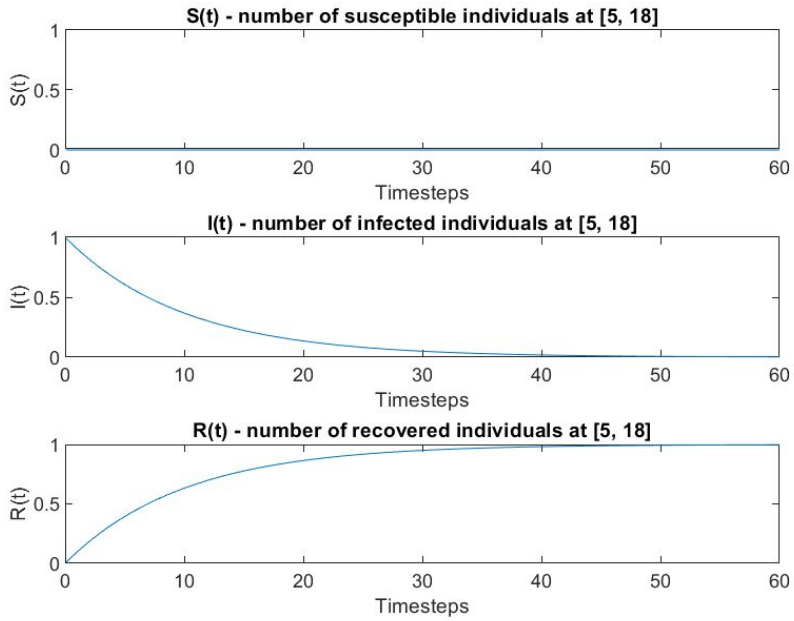


Figure 3. The SIR plots over time at (5,18) position

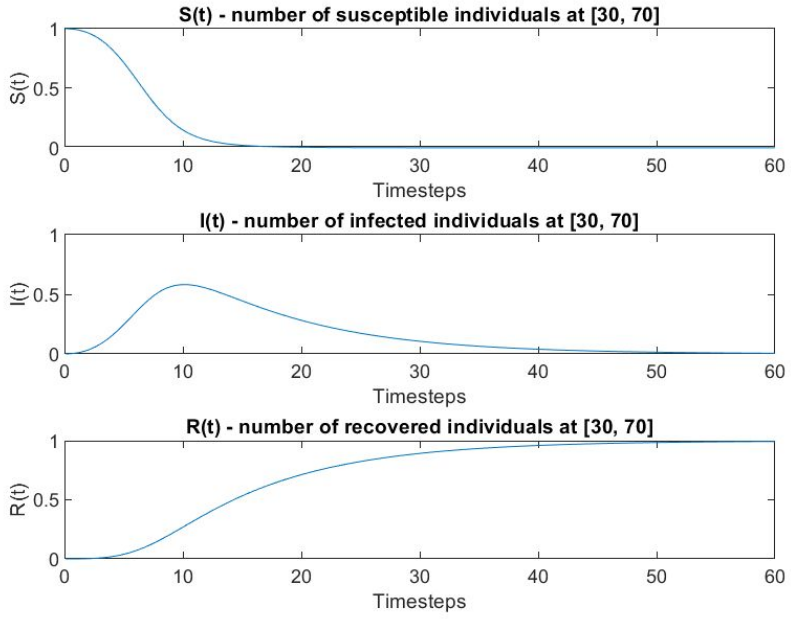


Figure 4. The SIR plots over time at (30, 70) position

3.8 Discussion

Based on the runtime results, the system solved by ode45 (Matlab built in function) takes less time to run than the system solved by RK4. With its own algorithm, ode45 solves the SIR system in 161 timesteps, while RK4 function solves the system in 601 timessteps. Therefore, the ode45 function is faster than the RK4 function.

According to figure 3, the (5,18) position in the grid is one of the sources of infection since all individuals are infected at the start time, and this number decreases over time. At this position, there is no data for the susceptible population. The number of recovered individuals starts at 0 and increases over time, the long term behavior is opposite with the number of infected individuals.

According to figure 2 and 4, the (30, 70) position is closer to the source of infection than the (1,1) position. The number of susceptible individuals at (30, 70) starts decreasing at the beginning; the number of infected individuals reaches the maximum at the time steps 10. At the same time, the number of susceptible individuals at (30, 70) starts decreasing at ~12th time steps; the number of infected individuals reaches the maximum at the time steps 22-23. Therefore, individuals at (30, 70) position are all recovered sooner than the ones at (1, 1), since the disease spreads to the (30, 70) position first before getting to the (1, 1) position).

4. Conclusion

Principal components analysis (PCA) and spatial SIR are two of methods used to study the spread of disease. The PCA is a useful method in analyzing complex data sets, projecting the data and figuring out the correlation between variables. Besides, the spatial SIR model helps us to visualize the spread of disease and track the number of S, I, R over

time at local positions. The spread of disease is affected by a variety of factors; these models cannot predict exactly what will happen in the real world. However, these methods help us identify and calibrate our actions, and we can have a plan to prevent the worst outcome happening.

References

1. Jaadi, Zakaria. "A Step-By-Step Explanation Of Principal Component Analysis". *Built In*, https://builtin.com/data-science/step-step-explanation-principal-component-analysis.

2. Jawed, Mohammad K., et al. “Final project of M20 Introduction to Computer Programming ", University of California, Los Angeles, 2020.