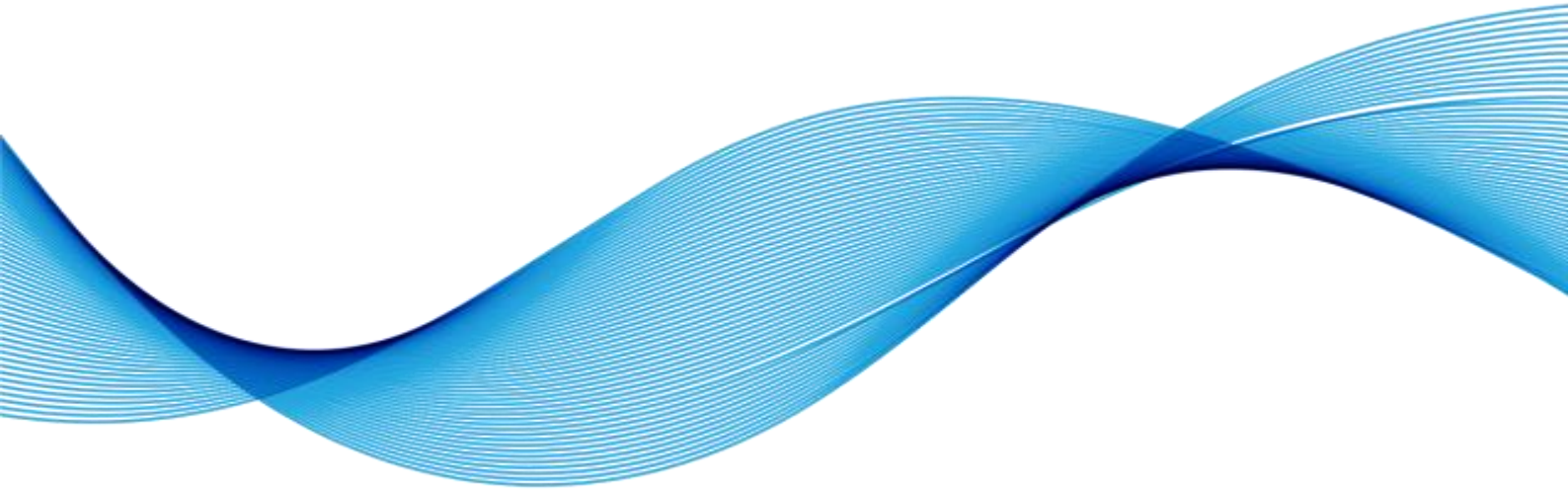
System Modeling and Simulation



Tasoulas Theofanis

January 2019

CONTENTS

INTRODUCTION

System simulation and modeling is the process of creating a conceptual or physical model regarding the operation of a system (active system or under construction), as well as finding its evolution equation. This helps to understand how it works and make decisions on it. Describing the operation of the systems is done by introducing appropriate sizes, where they can be quantified. These are called constitutive variables, which count a quantity and uniquely correspond to a system parameter. For example, in order to predict the moon's trajectory, it is sufficient to know its location and its speed per time. Therefore, two constitutive variables are needed, one corresponding to the position at a time and one corresponding to the speed per time of the moon.

EXPERIMENT 1 - VIRUS TRANSMISSION

Suppose a virus, which can be transmitted to a network with unique hosts, where N is the total number of nodes, S the uninfected nodes and I the infected nodes (N=S+I). At a given time t ≥ 0, the total of N network is divided into infected I and uninfected S, represented by I (t) and S (t), respectively. I (t) is the number of nodes infected at time t and S (t) is the number of nodes not infected at time t. Time can take any logical unit of measure (seconds, minutes, hours, etc.). For time t = 0 it is assumed that (S (0), I (0)) = (S_0, I_0), where I_0 = 1 is the initially infected number. Due to the random propagation behavior of the virus for t> 0, S (t) and I (t) are random variables. The infected nodes propagate their infection by removing contaminated packets into other randomly selected fixed rate B nodes, where 1 ≥ B ≥ 0. For all times t ≥ 0, it is confirmed that I (t) + S (t) = N. When an infected node attempts to propagate the virus in a randomized B series, it is assumed that its targets are selected with a uniform distribution. The experiment ends when all nodes are infected. Therefore, the differential equation that solves the problem is:

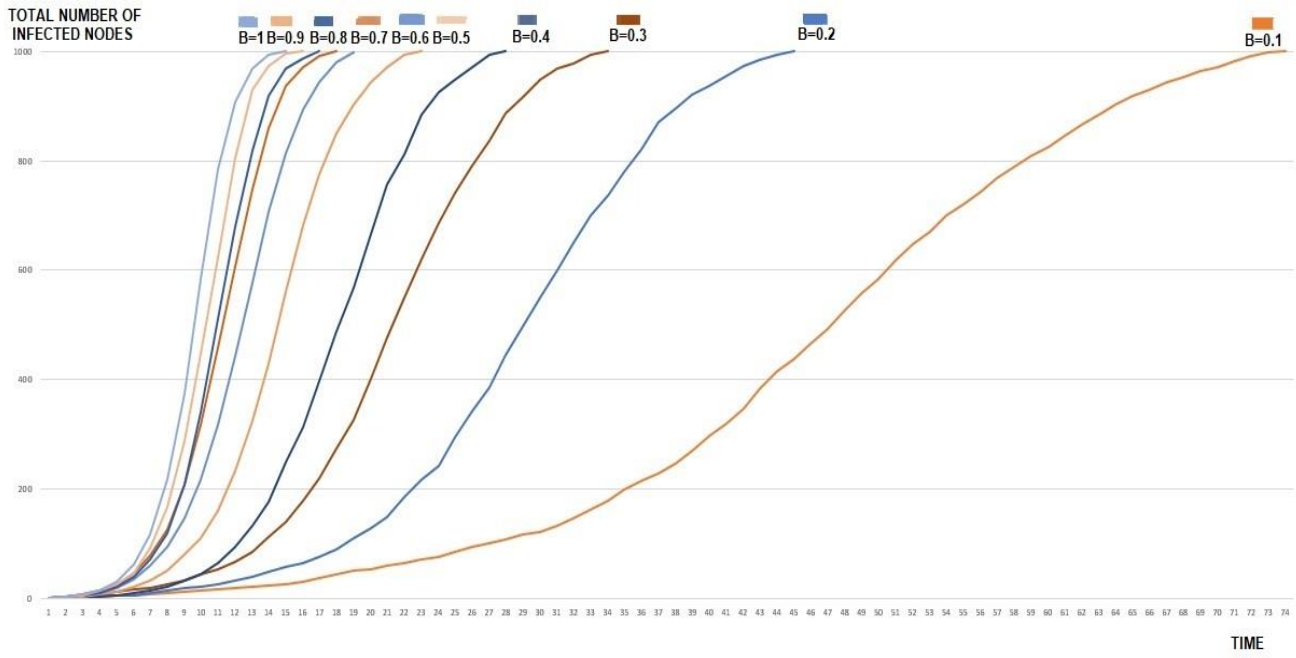
$$\frac{dI}{dt} = \beta \cdot I\left(N - I\right)$$

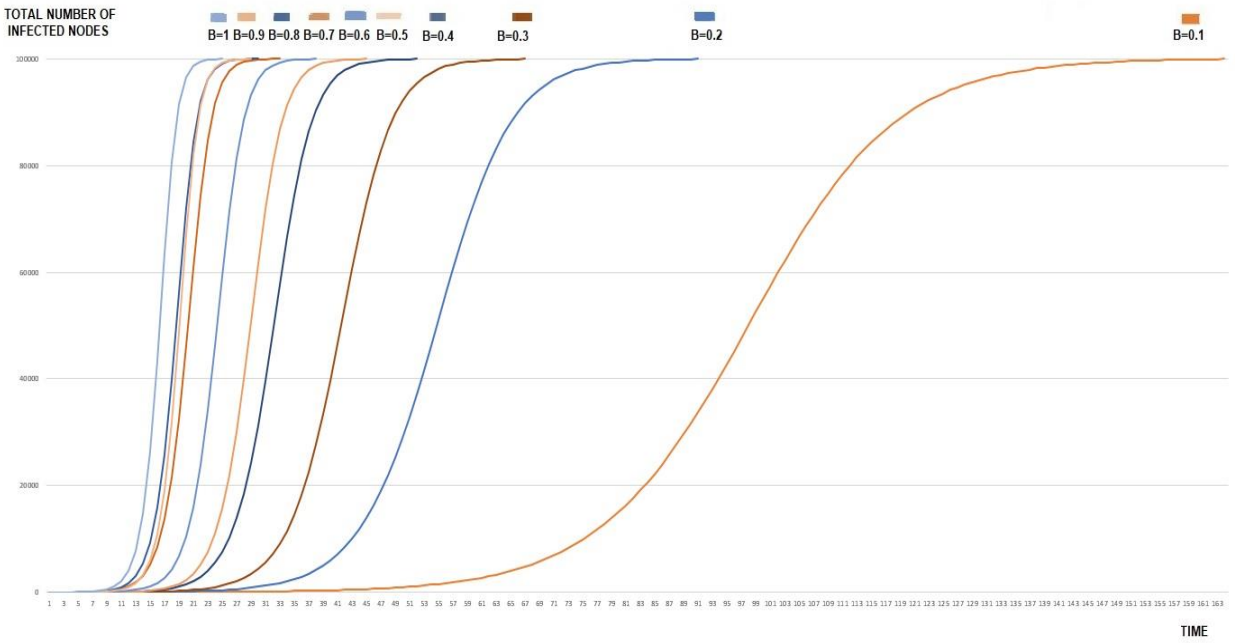
The above problem is represented by the following code, written in C language.

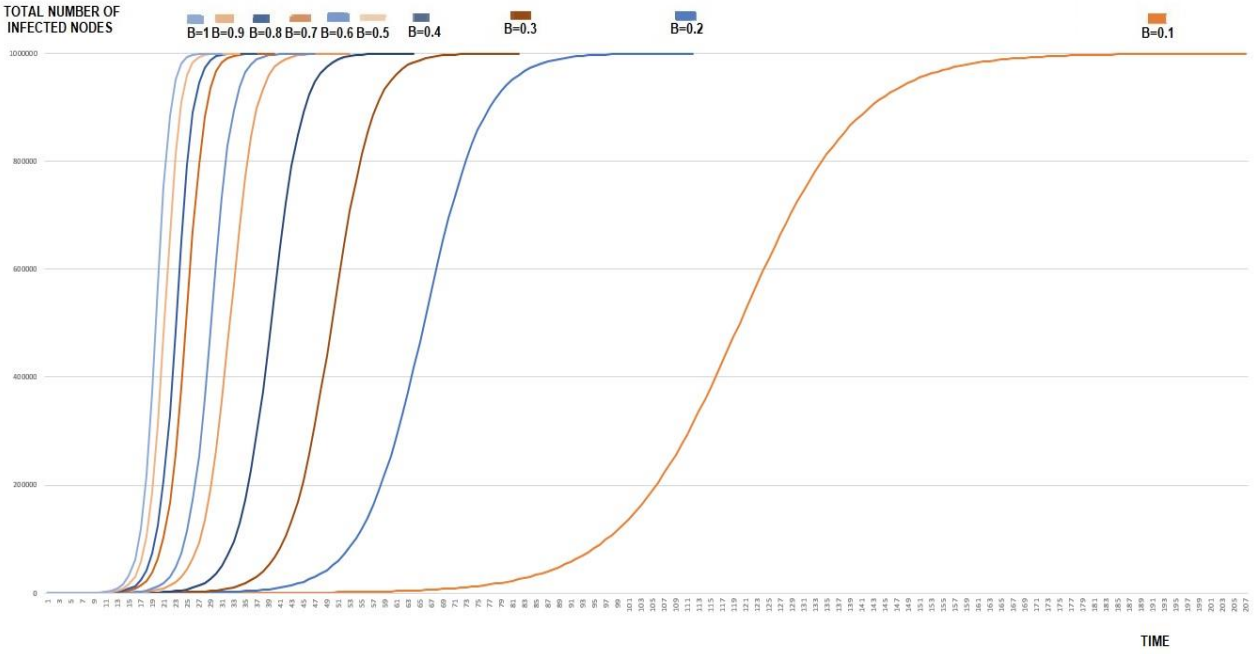
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NODES 1000000
#define INFECTED 1
#define HEALTHY 0

int main(void)
{
srand(time(NULL));                                  // Randomize the time
FILE *FilePointer = fopen("Result.csv", "w");       // Make a file, to write results on it
float B = 1;                                        // Randomness variable
for (B = 1; B<=10; B++){
   int TotalInfected=1;                             // Num of infected
   int network[NODES]={HEALTHY};                    // All network nodes are healthy
   network[rand() % 1000000] = INFECTED;            // One  network node is infected (radomly)
   int j = 1;                                       // Variable used on the "while" loop
   int time = 0;
   int random;
   float rnd;
   printf("\n\nRadomness B = %0.1f\n", B/10);
   int NITT = 0;                                    // Number of infected this time

   while (TotalInfected < NODES && TotalInfected > 0){
      for (int i = 1; i<=j; i++){
         random = rand() % 1000001;                 // Choose a random network
         rnd = (float) rand() / (float) RAND_MAX; // Choose a random number
         if ( rnd  <=  B/10){                  // If the random number is less than B (where 0.1 < B < 1)
            if (network[random] == HEALTHY){  // then, if the random network is healthy
               network[random] = INFECTED;       // then make it infected
               TotalInfected++;                  // also add 1 to the Number of infected Networks
```

```
                NITT++;                 // also add 1 to the Number of infected Networks this time
            }
        }
    }
    j = TotalInfected;
    time++;
    printf("Time: %d \nNum of Infected: %d \n", time, TotalInfected);
    fprintf(FilePointer, "%d; %d; %d; \n", time, TotalInfected, NITT);
    NITT = 1;                           // re initialize Number of infected Networks this time
  }
}
fclose(FilePointer);
return 0;
}
```

By running the code, the values of the total infected nodes I per time t and the nodes infected per time t for the different constants B = {0.1, 0.2, ..., 1} are extracted into a .csv file. If the values per constant B are separated, the graphs of the total number of infected nodes N per time t can be generated with a number of nodes N equal to 1,000, 100,000, 1,000,000 respectively:

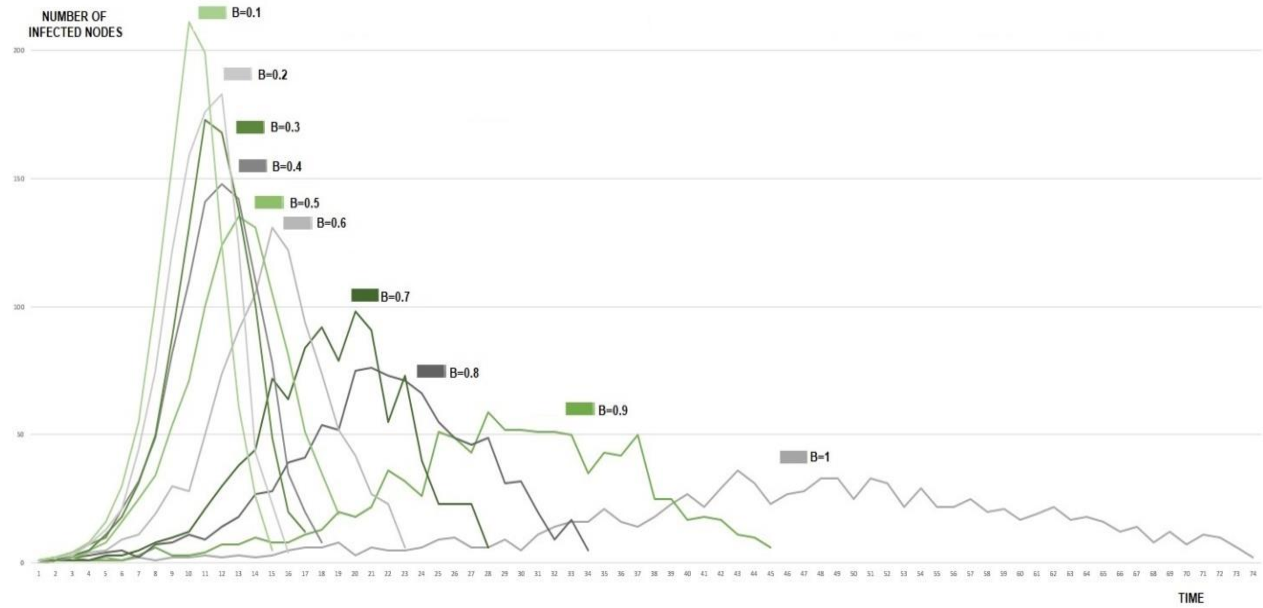


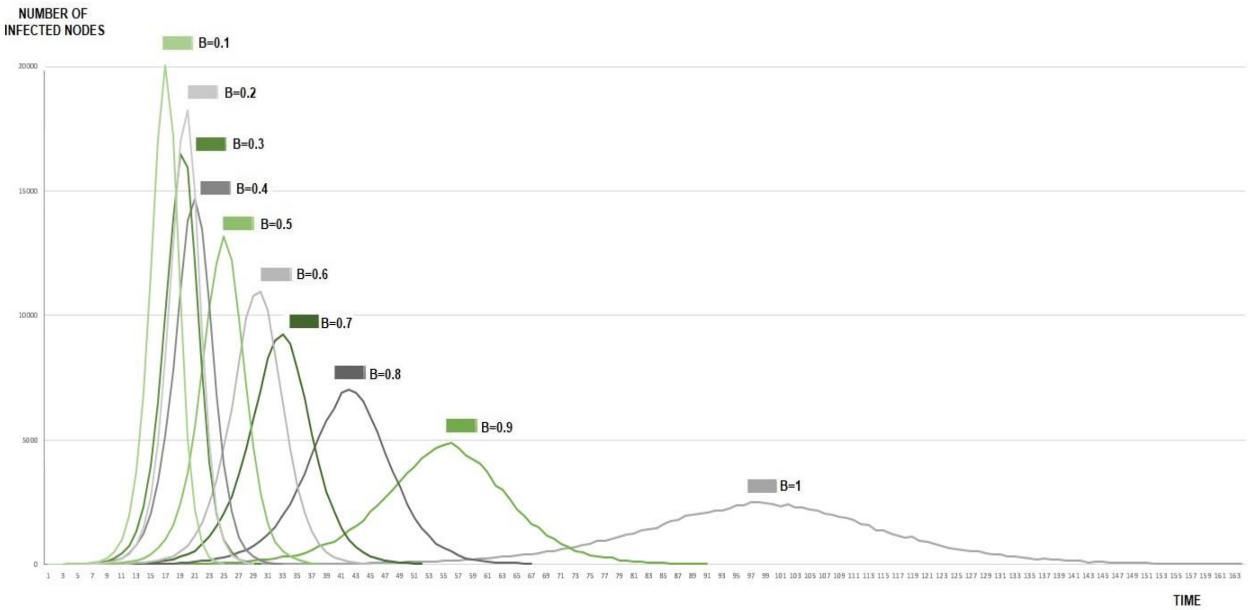


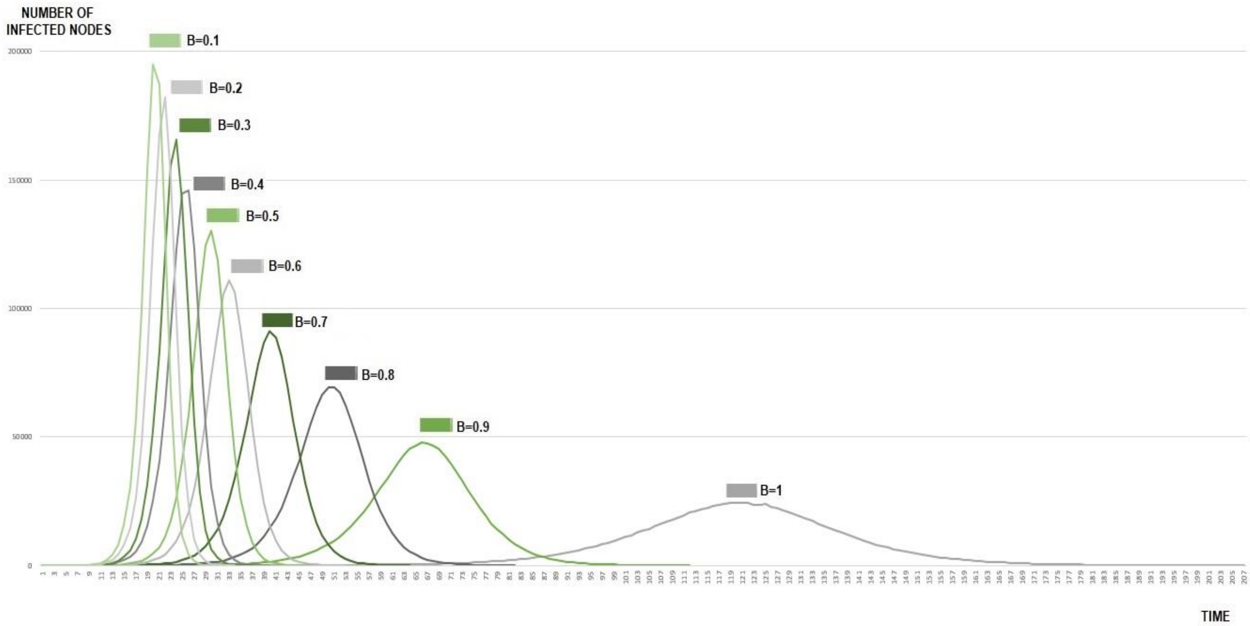


Based on the graphs, the number of infected nodes, regardless of the B rate, initially for a very short time t does not increase very much, as there are not many infected nodes yet. However, soon enough the nodes will begin to infect at a steep rate. This is because a set of nodes (about 2.5% to 5% of N) is infected, large enough to infect all other healthy nodes. Finally, when almost all the nodes are infected, the healthy ones are very few (about 2% to 4% of N) and consequently, it is more likely that a contaminated node will receive the infected package than a healthy node (the receiving is random). That's why the rate of infection is decreasing. Therefore, it is very logical to create a sigmoid graph.

Below, the graphs of nodes infected at time I (t), with node ns of 1,000, 100,000 and 1,000,000 respectively, are shown.


NUMBER OF
INFECTED NODES
B=0.1
B=0.2
B=0.3
B=0.4
B=0.5
B=0.6
B=0.7
B=0.8
B=0.9
B=1
TIME


NUMBER OF
INFECTED NODES
B=0.1
B=0.2
B=0.3
B=0.4
B=0.5
B=0.6
B=0.7
B=0.8
B=0.9
B=1
TIME


NUMBER OF
INFECTED NODES
B=0.1
B=0.2
B=0.3
B=0.4
B=0.5
B=0.6
B=0.7
B=0.8
B=0.9
B=1
TIME

In the beginning, the number of nodes infected per time I (t) (about 27% -32% of t time) is small, as few are the already infected I and it is not that easy to spread a few nodes to the virus. But then, I (t) sharply increases for a period (about 26% -32% of time t). After that, the growth rate decreases in the same way as the previous rate (about 26% -32% of time t), which forms a “bell” in the graph. Finally, I (t) decreases slowly (about 7% -12% of time t). This is reasonable for the same reasons as the previous case

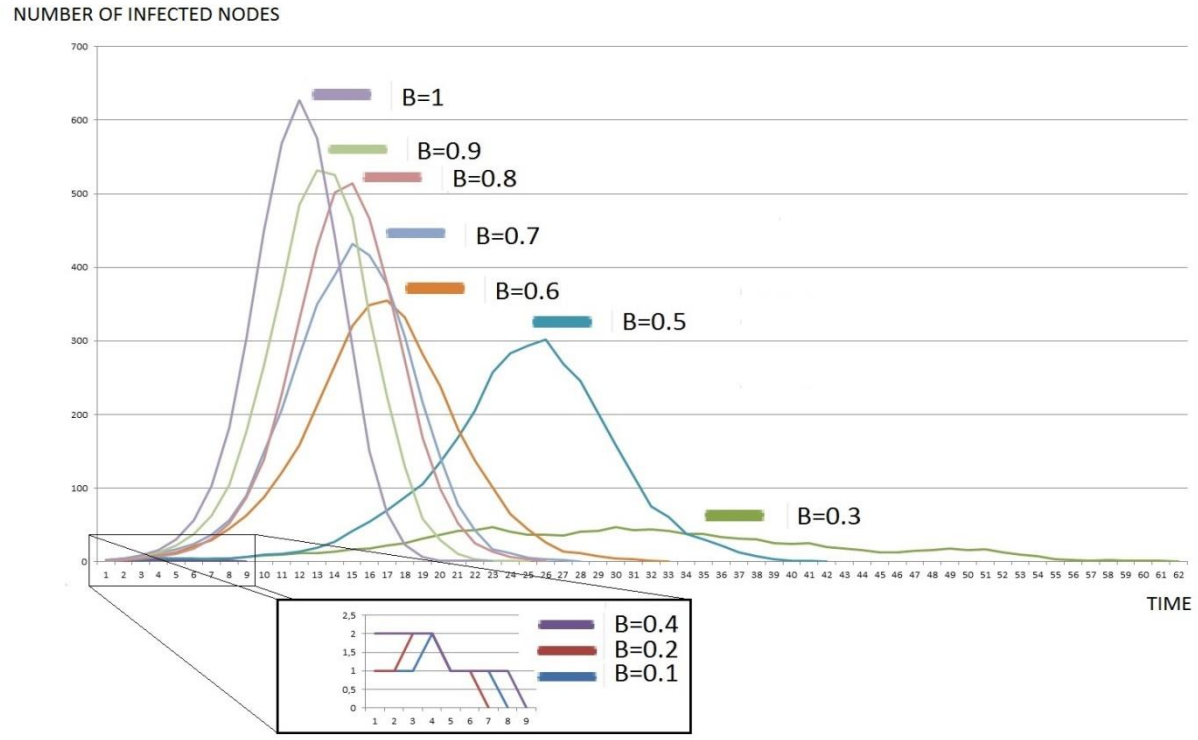
EXPERIMENT 2 - VIRUS AND VACCINES TRANSMISSION

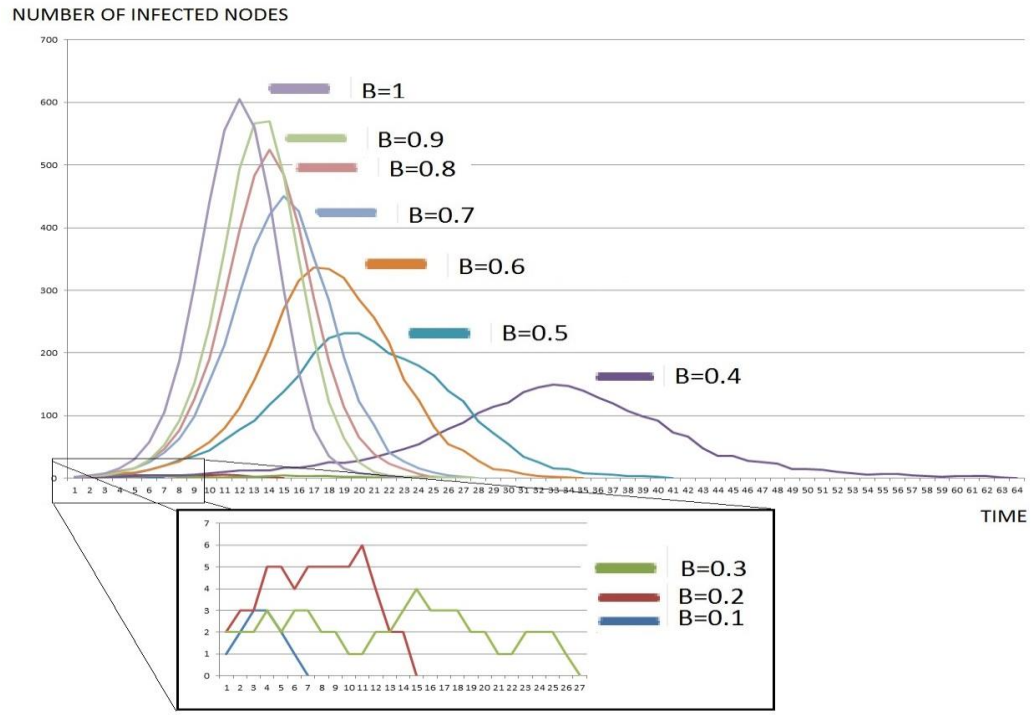
Now the complexity of the experiment increases with the addition of the "human factor". Now, each infected node will be immunized after t = 5 and it won't be able to be infected again (only one node can be immuned at a time). The experiment ends when all nodes are infected or immunized. This is the implementation code.

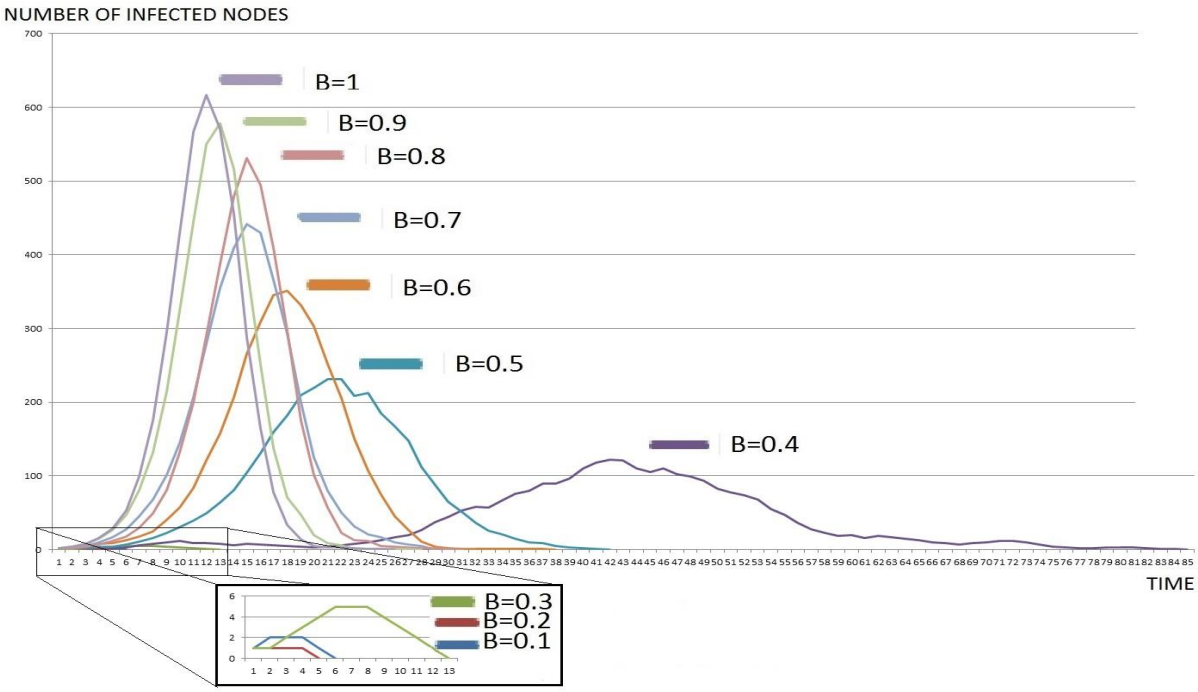
```
typedef struct NODESTRUCT{
int state;
int healingTime;
} Node;

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define NODES 1000000
#define HEALTHY  0
#define INFECTED 1
#define IMMUNED  2

int main(void){
srand(time(NULL));                                  // Randomize the time
FILE *FilePointer = fopen("Result2.csv", "w");   // Make a file, to write results on it
float B = 1;                                        // Randomness variable

for (B = 1; B<=10; B++){
   Node network[NODES];
   for (int i = 0; i<NODES; i++){                   // Initialize
      network[i].state = HEALTHY;                   // All network nodes are healthy and have healing
time=5
      network[i].healingTime = 5;
   }
   network[rand() % NODES].state = INFECTED;// One  network node is infected (radomly)
   int TotalInfected=1;                             // Num of Total infected
   int TotalImmunded=0;                             // Num of immuned
   int j = 1;                                       // Variable used on the "while" loop
   int time = 0;
   int random;
   float rnd;
   printf("\n\nRadomness B = %0.1f\n", B/10);
   int NITT = 0;                                    // Number of infected this time

   while (TotalInfected < NODES && TotalInfected > 0){

      for (int i = 1; i<=j; i++){
       random = rand() % 1000;                              // Choose a random network
            rnd = (float) rand() / (float) RAND_MAX;  // Choose a random number
            if ( rnd  <=  B/10){              // If the random number is less than B (where 0.1 < B < 1)
               if (network[random].state == HEALTHY){  // then, if the random network is healthy
                  network[random].state = INFECTED;      // then make it infected
                  TotalInfected++;          // also add 1 to the Number of infected Networks
                  NITT++;                   // also add 1 to the Number of infected Networks this time
```

```
                //if (TotalInfected==NODES){break;}  // If all networks get infected, exit the loop
            }
        }
    }
    for (int node=0; node <= NODES; node++){
      if (network[node].state == INFECTED){
        network[node].healingTime--;
        if(network[node].healingTime==0 ){
          network[node].state=IMMUNED;
          TotalImmunded++;
          TotalInfected--;
        }
      }
    }
    j = TotalInfected;
    time++;
    printf("Time: %d \nInfected: %d Immuned: %d\n", time, TotalInfected, TotalImmunded);
    fprintf(FilePointer, "%f; %d; %d;\n", B, TotalInfected, TotalImmunded);
    NITT = 1;                          // re initialize Number of infected Networks this time
  }
}
fclose(FilePointer);
return 0;}
```

Running the 2nd code results the values of the total infected per time t, for the different constants B = {0.1, 0.2, ..., 1} with nodes N = 1,000, N = 10,000 and N = 1,000,000, respectively.

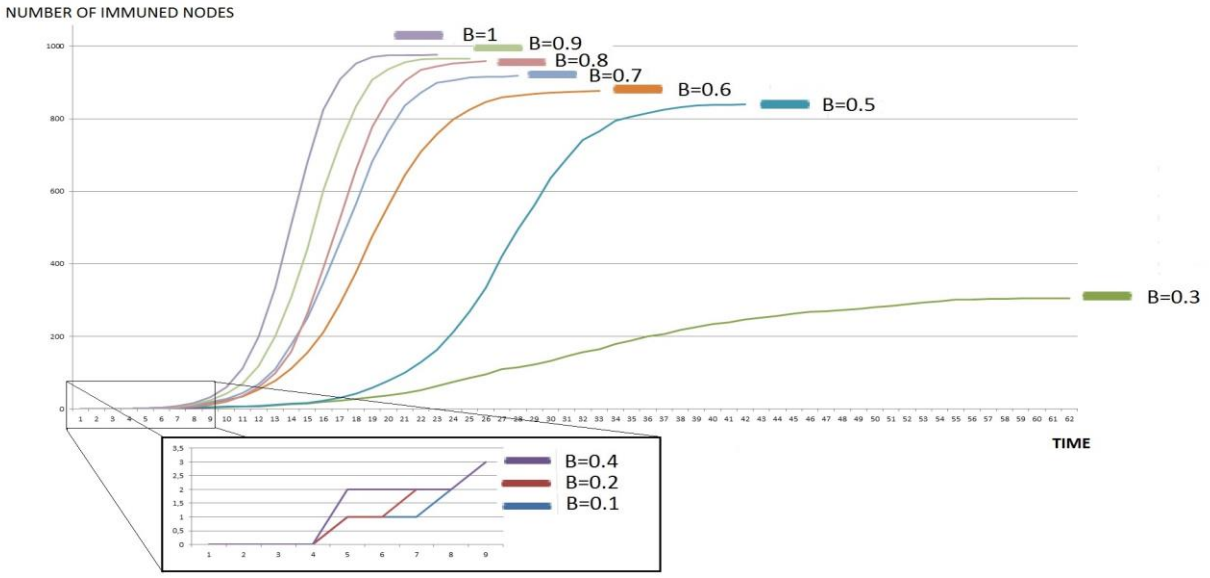


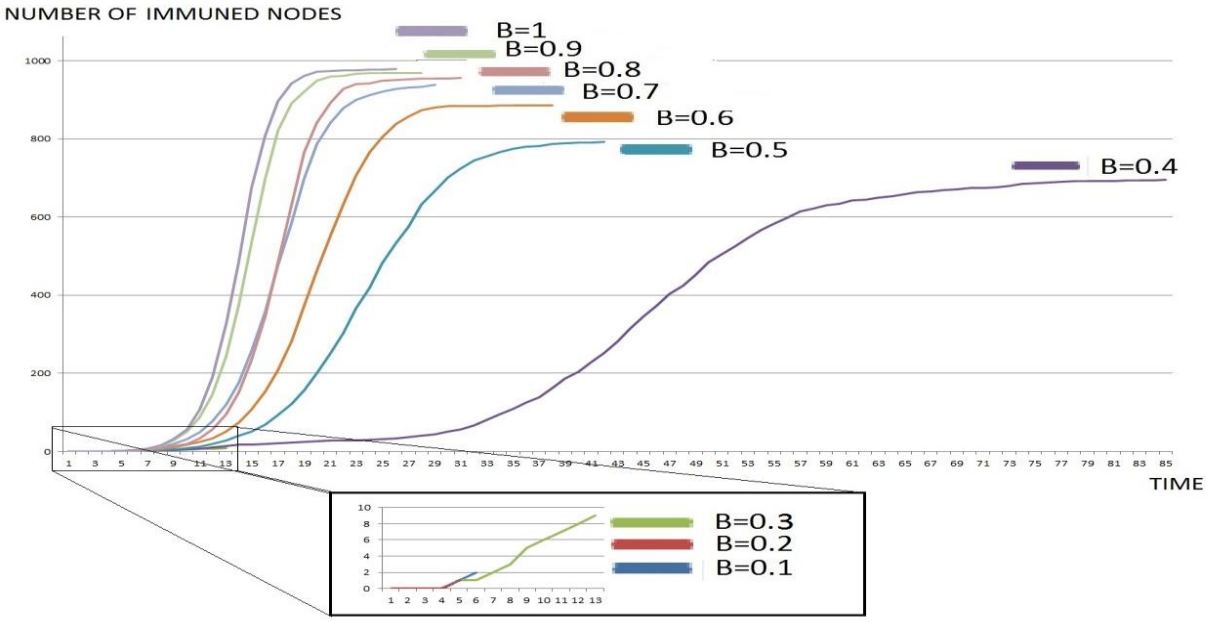


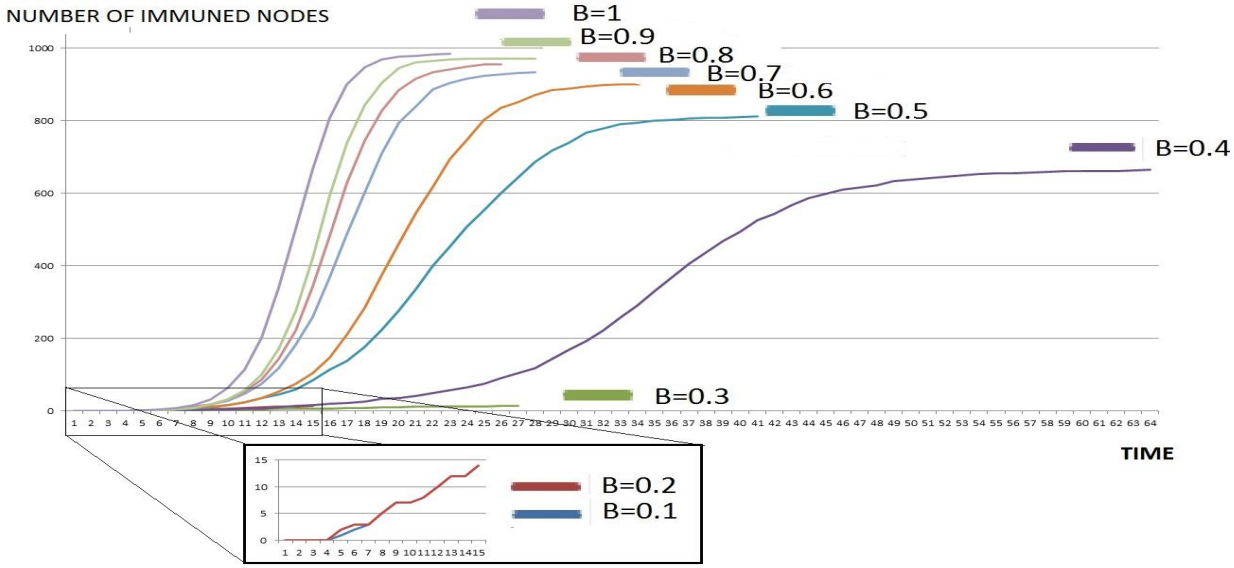


One observation of the above is that, the larger the B is, the larger the graphic “bells” are formed. That is because the probability of infection is great and eventually more nodes are infected. On the contrast, for small B, the virus does not spread and the infected nodes are already being immunized. This is why there are very small “bells” for B=0.1, 0.2 and 0.3 (the zoomed in the above images). In these, a few nodes are initially infected. But then, all of them are being treated and therefore the virus is eliminated in a short time. Remarkable is the rate of infection with the rate of immunization. The results are similar, at a similar time, regardless of the number of total nodes.

By the same procedure, the graphs of I (t) are plotted (total immunized nodes against time), with nodes n equal to 1,000, 100,000 and 1,000,000, respectively.







The totally infected nodes are represented in sigmoidal form for the same reasons of the previous experiment. The smaller the B, the fewer nodes need immunization. Under the current

circumstances, it seems that there is no way for all the nodes to be infected. Instead, the algorithm always ends with their healing.

EXPERIMENT 3 - INFORMATION TRANSMISSION

The next simulation refers to the transmission of information to a network. This time the network is represented as a two-dimensional array (like a cell with horizontal and vertical nodes). Firstly, a node is randomly selected and receives information (e.g. there is a fire nearby). At each time point, a random neighboring node is selected and receives the information from the previous node, located at a distance R (where R = 1, 2, ..., 10). The algorithm continues until all nodes receive the information. If a node receives the information more than once, nothing happens. The differential equation of the problem is given by the relation:

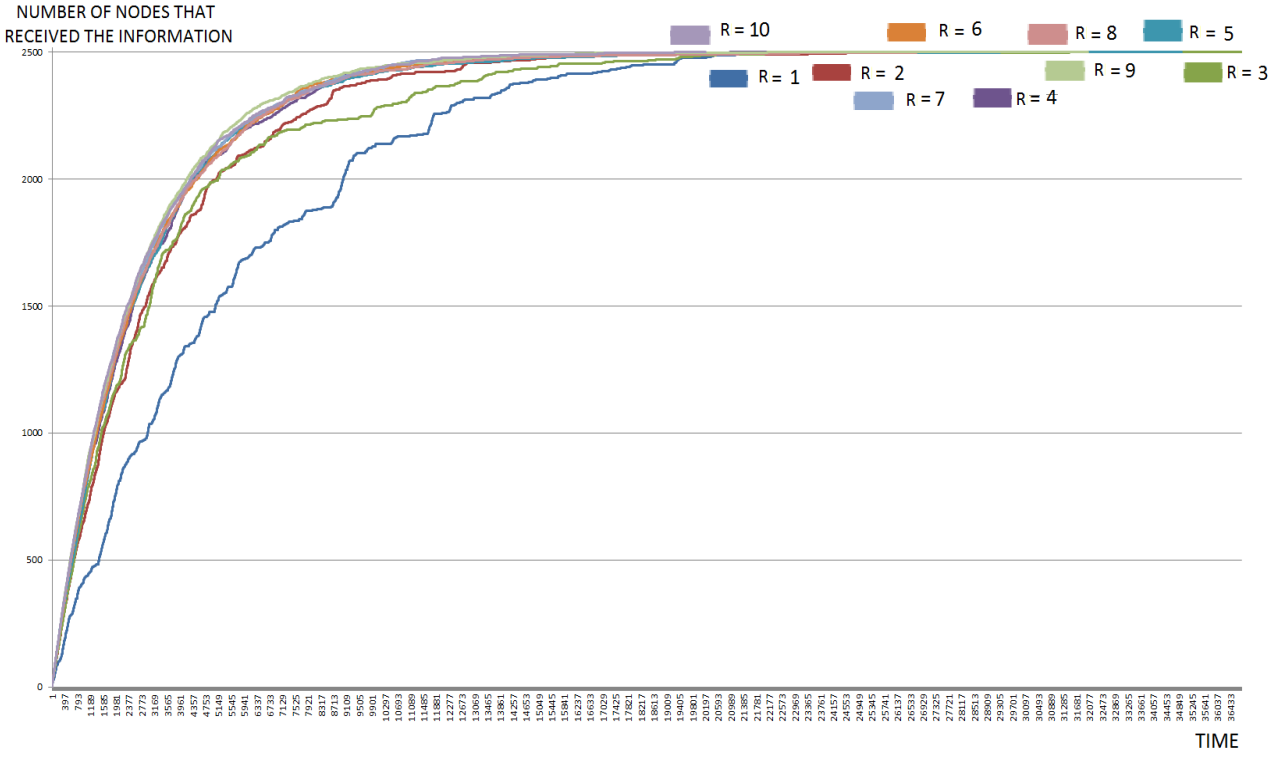
$$\frac{dI}{dt} = \beta \cdot (N - I)$$

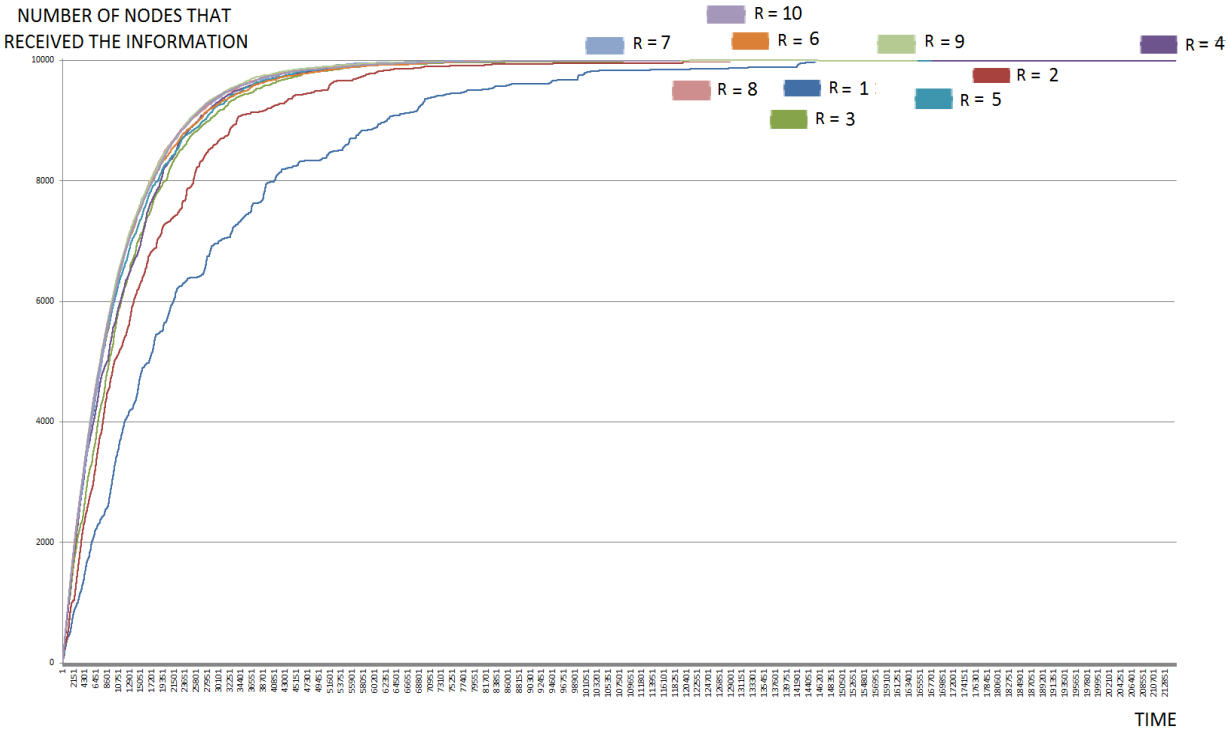
The following algorithm implements the transmission of information over a network.

```
typedef struct sensor{
int flag;
int radius;
float energy;
}Sensor;

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define NODES 10

int getRandomCoordinate(int coordinate, int radius, int coordinateMax);

int main(void){
srand(time(NULL));                    // Randomize the time
int time = 0;
float B = 1;                          // Randomness variable
int radius=1;                         // Radius variable
char radiusChar[6];                   // Just a variable to contain characters

for (radius=1; radius<=100; radius++){
   radius = radius+9;
   sprintf(radiusChar, "%d", radius);          // Convert int to char
   strcat(radiusChar, ".csv");                 // radiusChar + ".csv"
   FILE *FilePointer = fopen(radiusChar, "w");  // Make a file, to write results on it
   Sensor sensors[NODES][NODES]={0,-1,-1};
   sensors[rand()%NODES][rand()%NODES].flag = 1;
   int updatedSensor = 1;
   int currentSensorX = rand()%NODES;
   int randomSensorX = -1;
   int currentSensorY = rand()%NODES;
   int randomSensorY = -1;

   while(updatedSensor<NODES*NODES){
      do{
         randomSensorX=getRandomCoordinate(currentSensorX, radius, NODES);
         randomSensorY=getRandomCoordinate(currentSensorY, radius, NODES);
```

```
        }while(randomSensorX==currentSensorX);
        currentSensorY = randomSensorY;
        currentSensorX = randomSensorX;
        if(!sensors[currentSensorX][currentSensorY].flag){
           updatedSensor+=1;
           sensors[currentSensorX][currentSensorY].flag = 1;
        }
        time++;
        fprintf(FilePointer, "%d;\n", updatedSensor); //How many sensors got the message?
      }
   updatedSensor = 1;
   fclose(FilePointer);
}
   return 0;
}

int getRandomCoordinate(int coordinate, int radius, int coordinateMax){
   //one random value for one     coordination. x or y.
   int min = (coordinate-radius)<0?0:coordinate-radius;
   int max = (coordinate+radius)>coordinateMax?coordinateMax:coordinate+radius;
   return  min + rand()%((max-min)+1);
}
```

So, ten files are extracted, with the number of nodes that received the information per time for ten different R distances. The graphs below represent the above-mentioned values when the nodes are a total of 2500 and 10,000 respectively. Finally, note that the radius number is displayed in the order which the algorithm ended from left to right (i.e., the algorithm first ended for R = 10, then for R = 1, then for R = 2 and so on).





It is noted, that initially (approximately 5% to 10% of the transmissions are transmitted) the transmission rate is relatively equal to time. But lately the transmission rate is decreasing dramatically. This happens because in the beginning most nodes have not received the information, so it is easy to select a node (randomly) that has not received the information yet. But later, it is difficult to select the nodes that have not received the information, because they are few and the transmission of the information is very random. It also appears that, the smaller the radius R exists, the less the transmission rate of the information. However, the time at which all nodes will receive the information is independent of the distance R, which is unreasonable. The expected result would be, for a shorter distance R, the more time needed to complete the algorithm, as long as the ability to spread the information decreases. However, the randomness of transmission is unpredictable and independent of other factors, which verifies the present results. Last but not least, the less nodes the network has, the less time it takes to transmit the information to all the nodes.

EXPERIMENT 4 - SIMULATION OF EARTHQUAKE (OFC MODEL)

The fourth and final part refers to an earthquake simulation with the OFC simple model. Within a certain time t, due to the various movements of the earth and its containments (people, animals, etc.), each point of the ground delivers Fout power over time. When the force of this point exceeds a certain limit called Fcrit, then this point of the ground moves and exerts a force on its adjacent points. In the present case, the spot exerts a quarter of the power it has transmitted. Therefore, this creates an earthquake. The differential equation is:

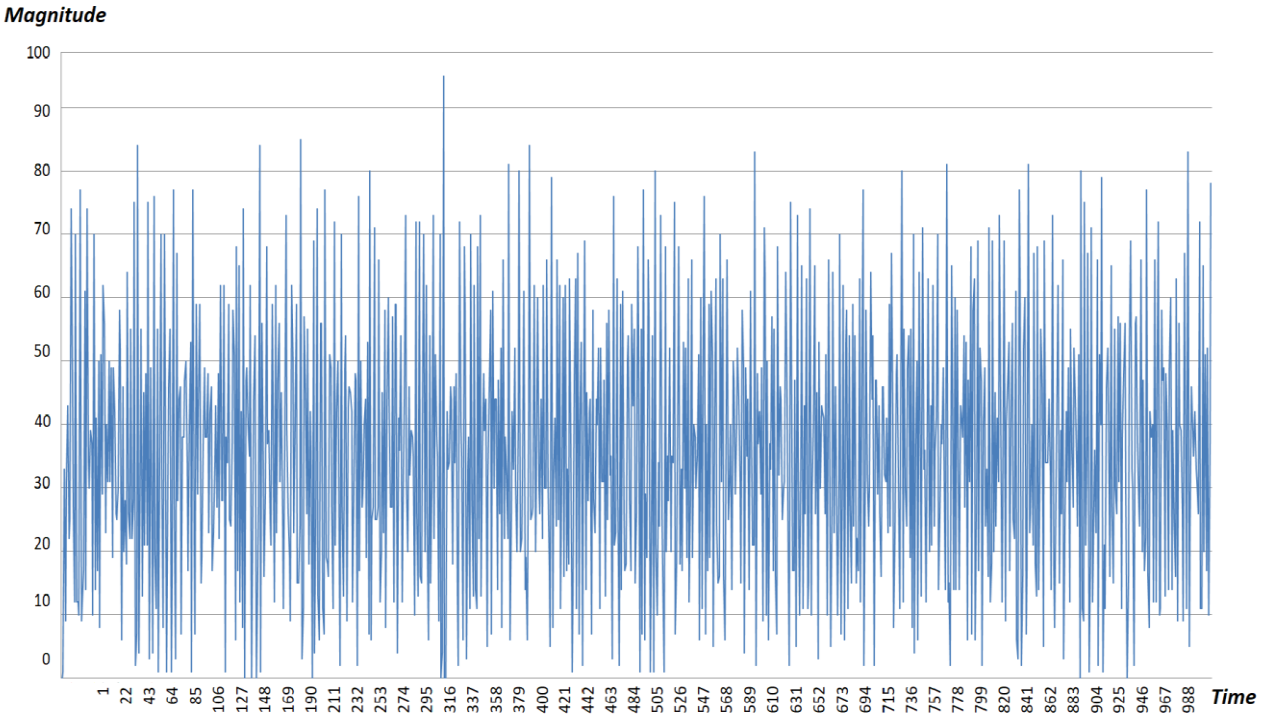
$$\frac{du}{dt} = t_{(external)} - t_{(friction)} + \int \frac{u-u_i}{|x-x_i|^a} du_i$$

u is the sliding, $t_{(external)}$ is the force of the rotary motion of the earth, $t_{(friction)}$ is the force of friction, t is the time and x is the distance. This can be implemented with the following code:

```
#include <stdio.h>
#include <stdlib.h>
#define Fcrit 0.4
#define Fout 0.01

int main(void){
float plate[10][10];
float helper_plate[10][10] = {0};
float earthquake[10][10] ={0};
int time = 0;
int isShaking = 0;
int i;
int j;
int magnitude=0;
FILE *FilePointer = fopen("Result.csv", "w");  // Make a file, to write results on it

for (i=0; i<=9; i++) {
        for (j=0; j<=9; j++){
                plate[i][j]=((float)rand()/(float)RAND_MAX)*Fcrit;
        }
}
while (time<=1000){
        isShaking=0;                    // No shaking
        for(i=0;i<=9;i++){              // Check if any plate is shaking
                for(j=0;j<=9;j++){
                        if(plate[i][j]>=Fcrit){// If the plate reaches Fcrit
                                isShaking=1; // there is at least a shaking plate
                                break;
                        }
                }
                if (isShaking){
                        break;
                }
        }
        for(i=0;i<=9;i++){
                for(j=0;j<=9;j++){
                        if(!isShaking){
                                plate[i][j]+=Fout;
```

```
                }
                else if(plate[i][j]>=Fcrit){ // Adjust the neighbor plates
                        if(i==0){               // 1st line
                                helper_plate[i+1][j] += 0.25 * plate[i][j];
                        }
                        else if(i==9){          // 10th line
                                helper_plate[i-1][j] += 0.25 * plate[i][j];
                        }
                        else{                   // Any other line
                                helper_plate[i+1][j] += 0.25 * plate[i][j];
                                helper_plate[i-1][j] += 0.25 * plate[i][j];
                        }
                        if(j==0){
                                helper_plate[i][j+1] += 0.25 * plate[i][j];
                        }
                        else if(j==9){
                                helper_plate[i][j-1] += 0.25 * plate[i][j];
                        }
                        else{
                                helper_plate[i][j+1] += 0.25 * plate[i][j];
                                helper_plate[i][j-1] += 0.25 * plate[i][j];
                        }
                        plate[i][j]=0;            // The plate lost its power
                        earthquake[i][j]=1;    // An earthquake began
                }
        }
    }
    if(isShaking){
        for(i=0;i<=9;i++){
                for(j=0;j<=9;j++){
                        plate[i][j]+=helper_plate[i][j];
                        helper_plate[i][j]=0;
                }
        }
    }
    if(!isShaking){
        magnitude=0;
        for(i=0;i<=9;i++){
                for(j=0;j<=9;j++){
                        if(earthquake[i][j]==1){
                                magnitude+=1;
                        }
                        earthquake[i][j]=0;   // An earthquake ended
                }
        }
        fprintf(FilePointer, "%d;\n", magnitude);
        time++;
    }
}
fclose(FilePointer);
return 0;
}
```

Running the above code produces a .csv file with the same graph as a seismogram:



SOURCES

KURT R. ROHLOFF - TAMER BAŞAR, April 2008, Deterministic and Stochastic Models for the Detection of Random Constant Scanning Worms

CALIBRATION OF DISCREET SYSTEMS, IONIAN UNIVERSITY DEPARTMENT OF INFORMATICS

MARKOS AVLONITIS - MARIA NEFELI NIKIFOROU - CHRYSA TSIVINTZELI, 2019, Notes - Course: Simulation and Modeling, IONIAN UNIVERSITY DEPARTMENT OF INFORMATICS