Modelization and Simulation of Biosystems

Pedro Bueso-Inchausti García

2020-1-13

Contents

1 Note to the reader

This report includes the exercises that were done for the subject of 'Modelization and Simulation of Biosystems' at the Master in Computational Biology from the Polytechnic University of Madrid.

First of all, one general explanation. As we were free to do whatever we wished, I opted for doing few exercises with a relatively high level of detail. I think this has allowed me to really understand the background theory behind each exercise and to explain the results and their implications coherently. In addition, I decided to think each problem not following exactly the guidelines that were given but trying to tell a logical story.

Each exercise has a similar structure. I start by briefly explaining the topic that lies beneath the exercise; I feel like this is important so to put the reader into context. Then, I include the functions that will be used during the exercise. Eventually I show the actual resolution, together with the results' interpretation.

The topics I have decided to cover are the following. 'Difference equation analysis' was the first subject of the course and, as many other topics used the mechanisms learned in it, I thought it was necessary to consolidate such knowledge. 'Infectious diseases and epidemic spread' is definitely an appealing topic; even people who find biology boring is somehow intrigued by how epidemics appear and evolve. 'Cellular automata' somehow reminded me of last year's internship; I worked with Agent Based Models and the way of working was quite alike. In addition, I had always found models like 'Game of Life' tremendously interesting, so I wanted to go deeper into this field. As for 'Networks', it was the paper we read about the six degrees of separation that increased my interest in this topic.

2 Difference equation analysis

Dynamic equations are those that change over time. In differential equations, time is considered to be continuous, so the equations are related with their derivatives. In difference equations, time is considered to be discrete, so the equations recursively define a sequence of values dependent from previous ones.

Dynamic equations are characterized by their order and their linearity. In differential equations, the order equals the higher derivative in the equation. In difference equations, the order is given by the time lapse. First order difference equations, $x_{t+1} = F(x_t)$, arise in many contexts in the biological, economic and social sciences. Such equations, even though simple and deterministic, can exhibit a surprising array of dynamical behavior, from stable points, to bifurcating hierarchies of stable cycles, to apparently random fluctuations.

2.1 Study of a difference equation

In this exercise, I will be studying the cubic difference equation, which is of the form $F(x) = rx^2(1-x)$.

When studying an equation, it is important to first visualize it. The graphical representation of the cubic equation shows us two symmetric behaviors depending on the value of r. If $r < 0$, the curve takes negative values for $x < 1$ (approaching to $-\infty$ as x becomes more negative) and positive values for $x > 1$ (approaching to ∞ as values becomes more positive). There is a local maximum at $x = 0$, being $F(0) = 0$, and a local minimum at $x = 2/3$, being $F(2/3) = 4r/27$. If $r > 0$, the curve takes positive values for $x < 1$ (approaching to ∞ as x becomes more negative) and negative values for $x > 1$ (approaching to $-\infty$ as values becomes more positive). There is a local minimum at $x = 0$, being $F(0) = 0$, and a local maximum at $x = 2/3$, being $F(2/3) = 4r/27$. In both cases, the x axis is cut at $x = 1$ and touched at $x = 0$.

For the mathematical determination of the critical points, we followed this procedure:
$F(x) = rx^2 - rx^3; F'(x) = 2rx - 3rx^2; F''(x) = 2r - 6rx$
$F'(x) = 0 \rightarrow 2rx - 3rx^2 = 0 \rightarrow x(2r - 3rx) = 0$
First critical point: $x = 0$
First critical point nature: $F''(0) = 2r \rightarrow$ maximum for $r < 0$ and minimum for $r > 0$
Second critical point: $2r - 3rx = 0 \rightarrow 2 - 3x = 0 \rightarrow x = 2/3$
Second critical point nature: $F''(2/3) = -2r \rightarrow$ minimum for $r < 0$ and maximum for $r > 0$

The next step in our analysis is to determine the range of x and r. In order to do so, we will assume that x represents a population density, which means it cannot take negative values and the maximum value is 1.

For determining the range of x, we will study the behavior of the function in its intervals. For both scenarios, $r < 0$ or $r > 0$, the only possible range of x is $0 < x < 1$. Out of it, the function approaches $-\infty$, 0 or ∞.

For r<0				
Interval	rx^2	$1-x$	$x_{t+1} \rightarrow$	$x_{t+k} \rightarrow$
$x < 0$	Negative, increasing with abs value	Positive, increasing with abs value	$-\infty$	$-\infty$
$x = 0$	0	1	0	0
$0 < x < 1$	Negative, increasing with abs value	Positive, decreasing with abs value	?	?
$x = 1$	1	0	0	0
$x > 1$	Negative, increasing with abs value	Negative, increasing with abs value	∞	∞

For r>0				
Interval	rx^2	$1-x$	$x_{t+1} \rightarrow$	$x_{t+k} \rightarrow$
$x < 0$	Positive, increasing with abs value	Positive, increasing with abs value	∞	∞
$x = 0$	0	1	0	0
$0 < x < 1$	Positive, increasing with abs value	Positive, decreasing with abs value	?	?
$x = 1$	1	0	0	0
$x > 1$	Positive, increasing with abs value	Negative, increasing with abs value	$-\infty$	$-\infty$

For determining the range of r, we are going to study the functions behavior on the already defined range of x. For $0 < x < 1$, only positive values of r make sense. The lower and upper limits are the ones leading to the minimum and maximum possible values of F(x). At whatever value of x, $F(x) = 0$ for $r = 0$. At $x = 2/3$, $F(x) = 1$ for $r = 27/4 = 6.75$. Thus, the only possible range of r is $0 < r < 6.75$.

A fixed point is a point of the function domain that is mapped to itself by the function. Whenever a function reaches a fixed point, it remains constant as time passes. From now on, we will refer to the fixed points of the cubic equation as x^*. To calculate the fixed points, we just need to solve the equation $x_{t+1} = x_t$, which is equivalent to finding the intersection between our function and the diagonal function. We find one trivial fixed point at $x = 0$ and two non-trivial fixed points at $x^* = \frac{r+\sqrt{r^2-4r}}{2r}$ and $x^* = \frac{r-\sqrt{r^2-4r}}{2r}$ that only appear for $r^2 > 4r \rightarrow r > 4$.

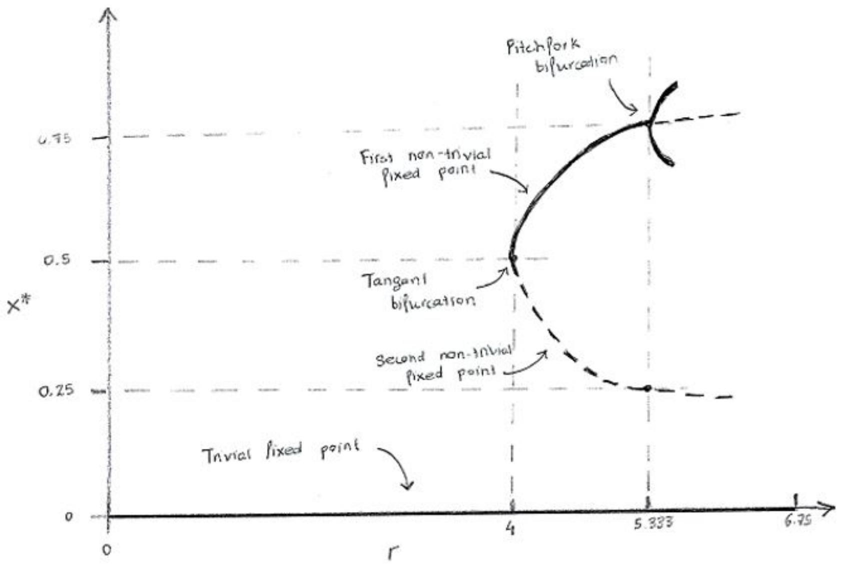
For the mathematical determination of the fixed points, we followed this procedure:
$x_{t+1} = x_t = x^* \rightarrow x^* = rx^{*2}(1 - x^*) \rightarrow rx^{*3} - rx^{*2} + x^* = 0$
Trivial fixed point: $x^* = 0$
First non-trivial fixed point: $x^* = \frac{r+\sqrt{r^2-4r}}{2r}$
Second non-trivial fixed point: $x^* = \frac{r-\sqrt{r^2-4r}}{2r}$

A fixed point is stable if the function tends to go back to it after a perturbation. We can check whether a fixed point is stable or unstable by looking at the slope of the tangent (λ). If it is between -1 and 1 (-45º and 45º), then the fixed point is locally stable because it attracts all the trajectories in the neighborhood; otherwise, the fixed point is unstable. We find that the trivial fixed point is stable in all the range of r, the first non-trivial fixed point is stable in $4 < r < 5.333$ and the second non-trivial fixed point is non-stable.

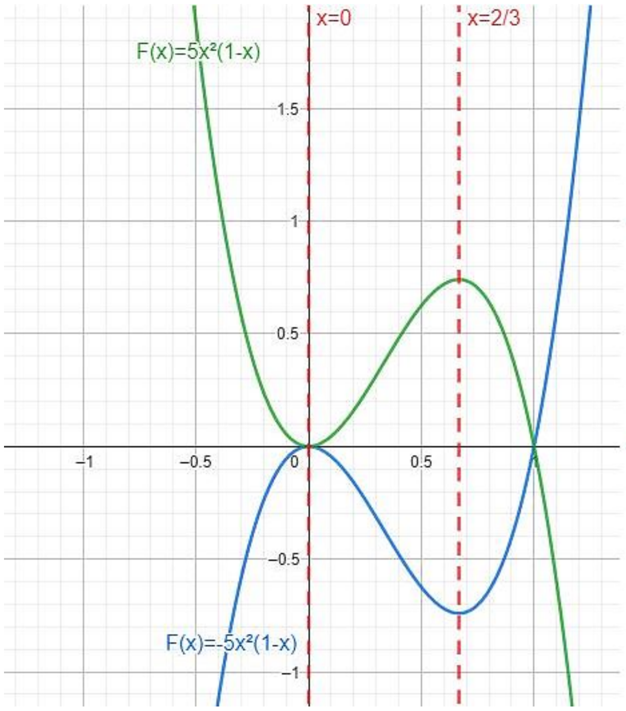
Both $x = 4$ and $x = 5.333$ are clear canditates for bifurcation events. At $x = 4$, there is only one fixed point ($x^* = 0.5$) which is stable and leads to two fixed points, one stable and one unstable. This is a clear example of tangent or $\lambda = +1$ bifurcation, which happens when the curve becomes tangent to the diagonal function. At $x = 5.333$, the previously created stable fixed point becomes unstable and a stable cycle of period 2 appears. This is a clear example of pitchfork or $\lambda = -1$ bifurcation, which happens when the curve becomes perpendicular to the diagonal function.

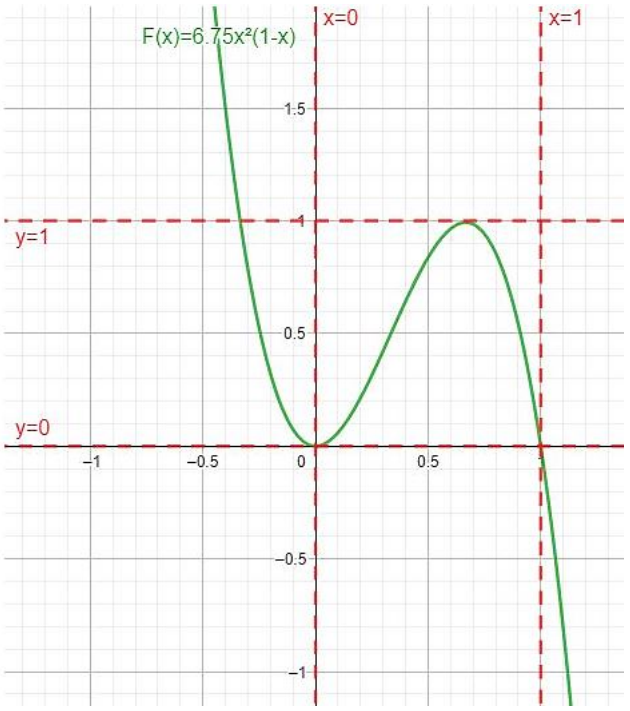
With what we have done so far, we have completed the bifurcation diagram of the cubic equation for $0 < r < 5.333$. The interval that lasts, $5.333 < r < 6.75$, should be analysed through higher iterations of the function. We will not complete such task but it is presumable that the behavior will become more complex.

This figure shows the incomplete bifurcation diagram.

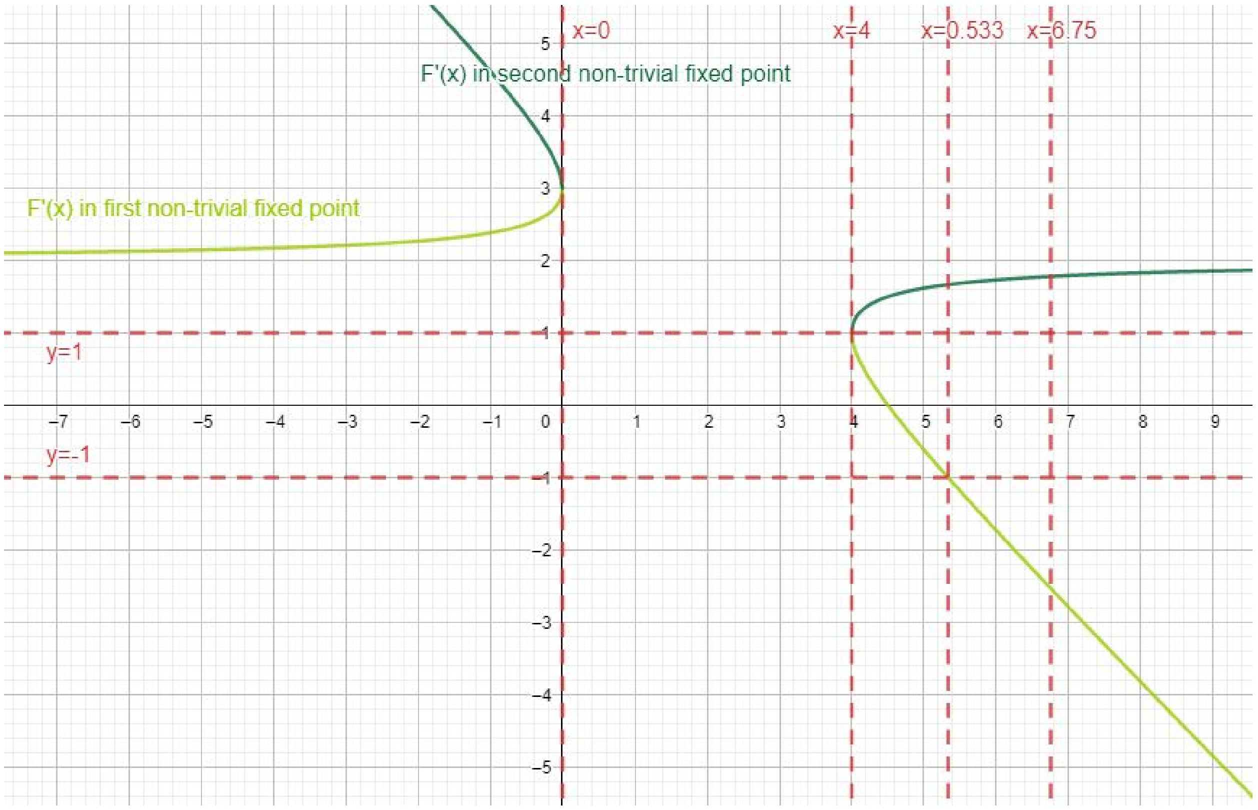


These figures show the basic features (left) and limits (right) of the cubic difference equation.

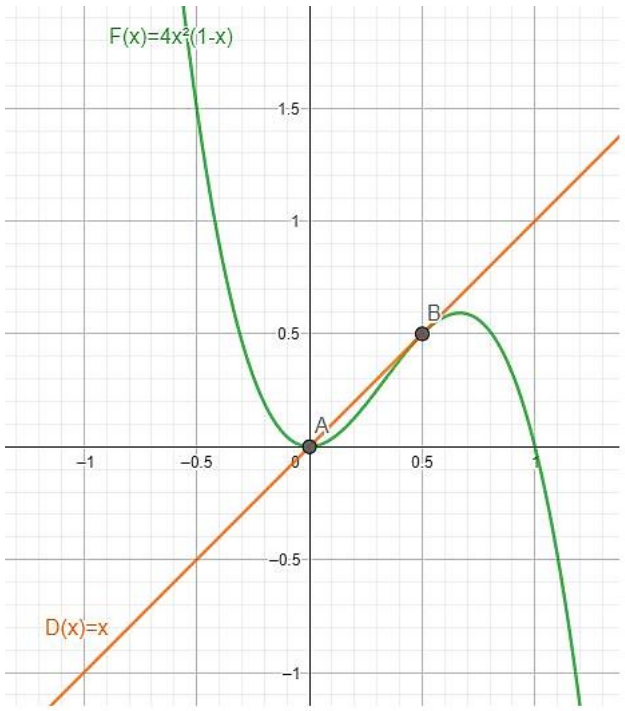


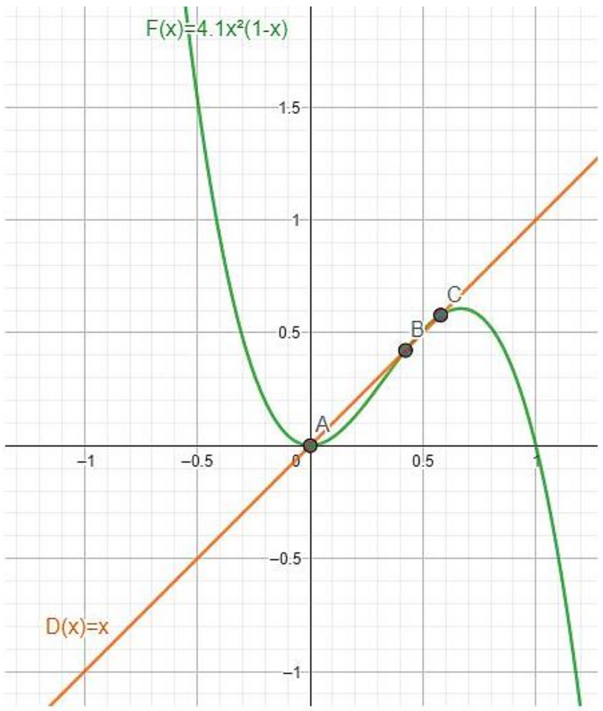


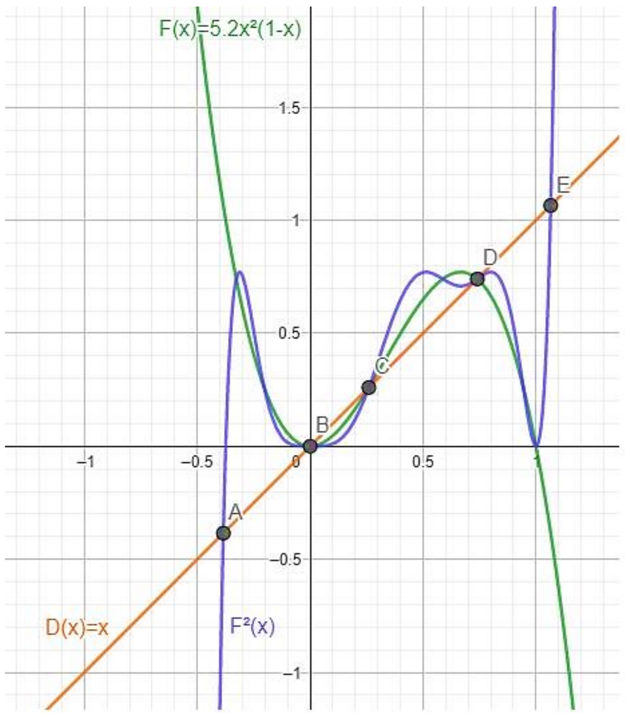
This figure shows the graphical stability determination for the non-trivial fixed points.

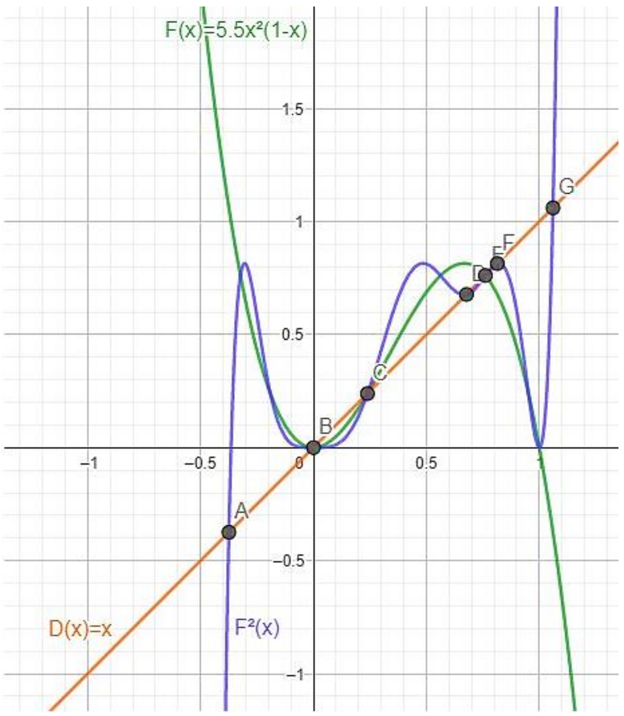


These figures show the two processes of bifurcation explained.









3 Infectious diseases and epidemic spread

Infectious diseases are tipycally studied through simplified compartmental models, where the population is divided into compartments and every individual in the same compartment is supposed to have the same characteristics. These models, which are implemented through ordinary differential equations (ODE), turn out to be very helpful when studying the properties and outcomes of an epidemic spread.

Before talking about the models themselves, let's put them into context.

An epidemic is basically the spread of an infection disease. However, it needs to fulfill two criteria: first, affect a large number of people in a given population; second, do so within a short period of time, usually less than two weeks. But how do epidemics occur? They are generally associated to scenarios where the host immunity to the pathogen is reduced, which can happen for many reasons (a change in the host population ecology, a genetic change in the pathogen, a new pathogen emergence…).

Some concepts that are closely related with the idea of epidemic, and which might be of interest, are the following. The epidemic threshold refers to the number or density of susceptible hosts that are required for an epidemic to occur. This quantity is tremendously useful as it is used to confirm the emergence (in order to set up appropriate control measures) and to assess the minimal levels of vaccination. The epidemic state refers the situation in which the infected population remains at a basal level. This is obviously not achieved when an infectious disease is eradicated. The epidemic incidence is its probability of ocurrence over a specific period of time. The maximum incidence refers to the maximum number of hosts that are infected at a given moment all through the epidemic.

Now that the context is clear, we can have a closer look into the models.

Although all of them are clear simplifications of the epidemic process, there are some that consider greater complexity by including intermediate states or allowing for cycles (going back to previous states). Down below I include a table with some popular epidemics models (S=Susceptible, E=Exposed, I=Infected, R=Recovered, β=Infection rate, α=Development rate, μ=Recovery rate, μ=Susceptibility rate).

Epidemic models			
Name	States	Transitions	Rates
SI	S, I	S $\rightarrow$ I	β
SIS	S, I	S $\rightarrow$ I $\rightarrow$ S	β, μ
SIR	S, I, R	S $\rightarrow$ I $\rightarrow$ R	β, μ
SIRS	S, I, R	S $\rightarrow$ I $\rightarrow$ R $\rightarrow$ S	β, μ, η
SEIR	S, E, I, R	S $\rightarrow$ E $\rightarrow$ I $\rightarrow$ R	β, α, μ
SEIRS	S, E, I, R	S $\rightarrow$ E $\rightarrow$ I $\rightarrow$ R $\rightarrow$ S	β, α, μ, η

3.1 Design a numerical simulation of the SIR model

In this exercise, we will perform a numeric simulation of the SIR (Susceptible-Infected-Recovered) model. In such epidemic model, susceptible individuals turn into infected individuals (with a certain infection rate, known as β) and these into recovered individuals (with a certain recovery rate, known as μ).

This function represents the SIR model as a set of differential equations.

```
SIR = function(time,state,parms)
{with(as.list(c(state,parms)),{dS=-beta*S*I; dI=beta*S*I-mu*I; dR=mu*I; list(c(dS,dI,dR))})}
```

This function simulates an epidemic process and plots its evolution.

```
simulate_SIR = function(S0,I0,R0,beta,mu,days)
{
  time = seq(0,days,by=0.01)
  matrix = ode(y=c(S=S0,I=I0,R=R0),times=time,func=SIR,parms=c(beta=beta,mu=mu))
  df = gather(as.data.frame(matrix),"S","I","R",key="State",value="Population")
  ggplot(df,aes(x=time,y=Population,color=State)) + geom_line(size=2) +
  xlab("Days") + scale_y_continuous(limits=c(0,1000)) +
  labs(caption=paste0("S0 = ",S0,", I0 = ",I0,", R0 = ",R0,", beta = ",beta,", mu = ",mu)) +
  theme_minimal() + theme(plot.caption=element_text(hjust=0.5))
}
```

These functions simulate an epidemic process and plot a phase space graph.

```
simulate_phase_space_SIR = function(S0,I0,R0,beta,mu,days)
{
  time = seq(0,days,by=0.01)
  matrix = ode(y=c(S=S0,I=I0,R=R0),times=time,func=SIR,parms=c(beta=beta,mu=mu))
  df = as.data.frame(matrix)
  df["Conditions"] = paste0("S0: ",S0,", I0: ",I0,", R0: ",R0)
  df["Parameters"] = paste0("beta: ",beta,", mu: ",mu)
  return(df)
}

plot_phase_space_SIR = function(df,y,ytag,type)
{
  plot = ggplot(df,aes(x=time,y=get(y),color=get(type))) + geom_path(size=1.5) +
  xlab("Days") + ylab(ytag) + scale_y_continuous(limits=c(0,1000)) +
  scale_color_grey(start=0.8,end=0.2) + theme_minimal() + theme(legend.position="none")
  return(plot)
}
```

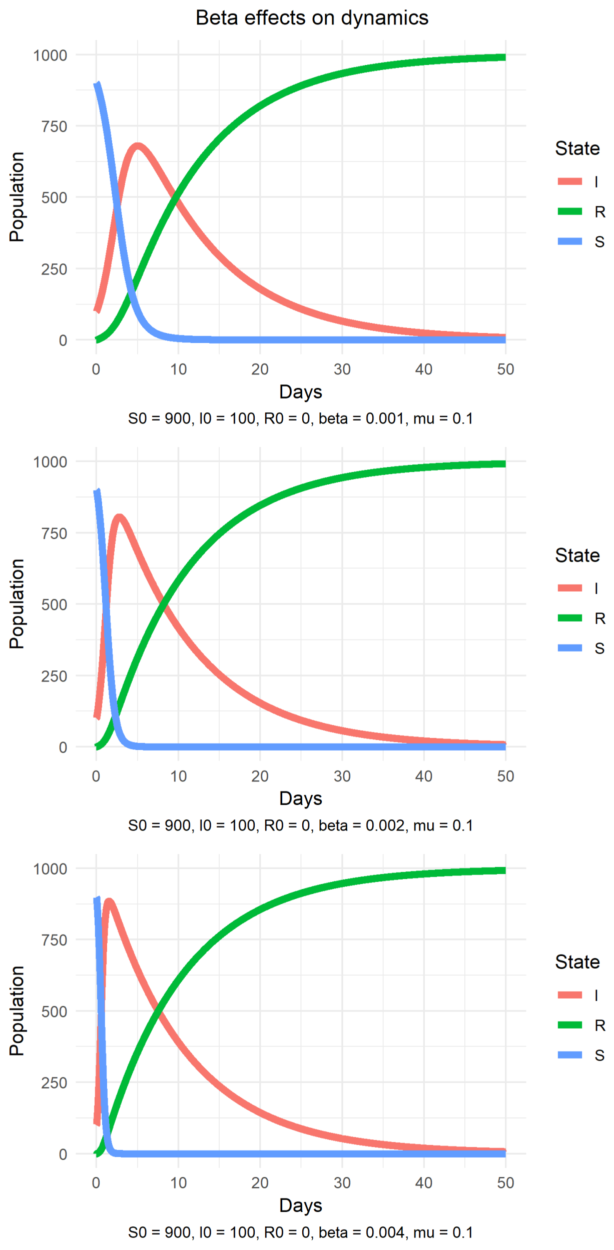
This function plots the evolution of a given parameter.

```
plot_evolution = function(df,x,y,color)
{
  plot = ggplot(evolution_df,aes(x=get(x),y=get(y))) + geom_line(size=2,color="grey20") +
  xlab(x) + ylab(y) + theme_minimal()
  return(plot)
}
```

The most important elements in this model are the ratios that define the transitions between different states. The infection ratio or β defines the transition between susceptible and infected individuals. The recovery ratio or μ defines the transition between infected and recovered individuals. Let's see their effects.

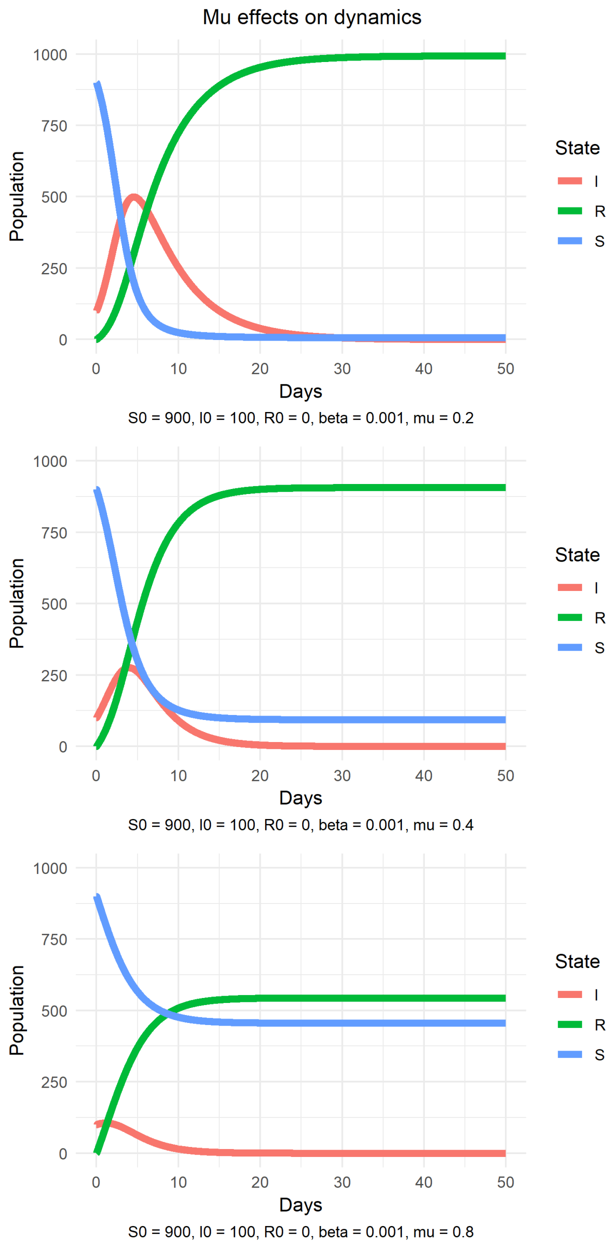
If we increase beta in relation to mu, susceptible individuals become infected individuals faster. This results in the infected state reaching high levels early. However, the fact that a recovery rate is implemented but a birth or reinfection rate is not, leads to the infected population eventually recovering (this is why the epidemic state is impossible in this model). The final scenario is that with all individuals recovered.

```
p1 = simulate_SIR(S0=900,I0=100,R0=0,beta=0.001,mu=0.1,day=50)
p2 = simulate_SIR(S0=900,I0=100,R0=0,beta=0.002,mu=0.1,day=50)
p3 = simulate_SIR(S0=900,I0=100,R0=0,beta=0.004,mu=0.1,day=50)
plot = arrangeGrob(p1,p2,p3,ncol=1,top="Beta effects on dynamics")
ggsave(filename="SIR1.png",plot=plot,width=5,height=10,path="Figures")
```



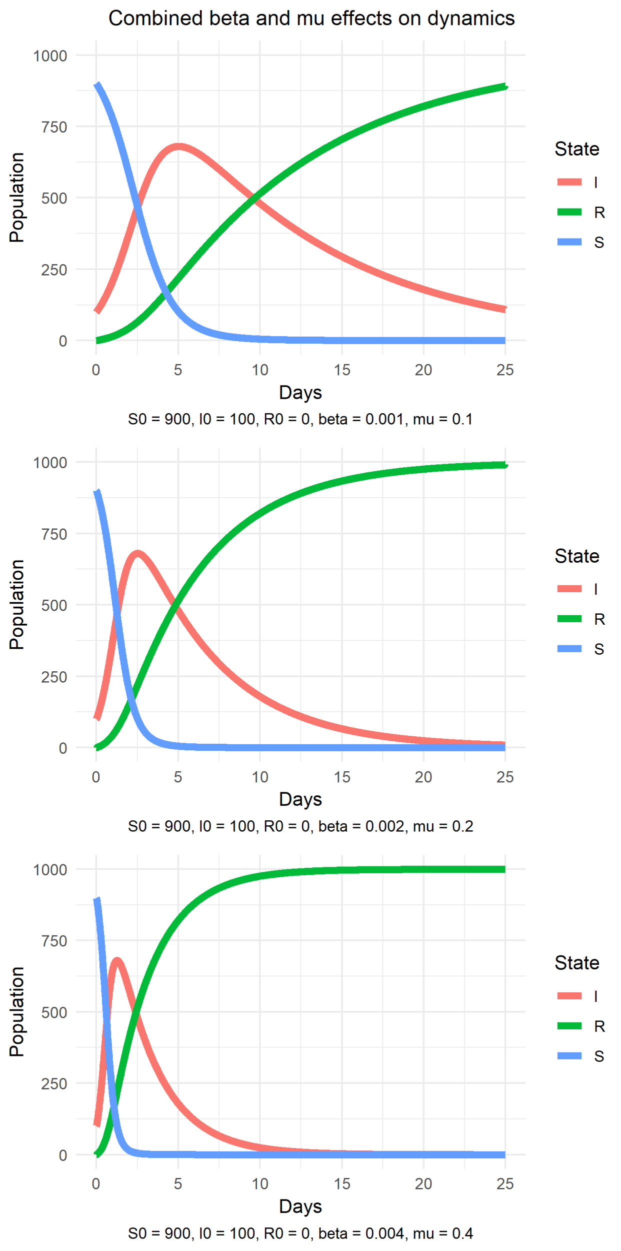
If we increase mu in relation to beta, infected individuals become recovered faster. This prevents the infection from reaching high levels at any point. The final scenario is that in which all individuals are susceptible or recovered. Note that, when infected individuals recover fast, this makes the disease less influential so the number of susceptible individuals ends up being higher.

```
p1 = simulate_SIR(S0=900,I0=100,R0=0,beta=0.001,mu=0.2,day=50)
p2 = simulate_SIR(S0=900,I0=100,R0=0,beta=0.001,mu=0.4,day=50)
p3 = simulate_SIR(S0=900,I0=100,R0=0,beta=0.001,mu=0.8,day=50)
plot = arrangeGrob(p1,p2,p3,ncol=1,top="Mu effects on dynamics")
ggsave(filename="SIR2.png",plot=plot,width=5,height=10,path="Figures")
```



If we increase both parameters in the same proportions, the dynamics stay the same but accelerate.

```r
p1 = simulate_SIR(S0=900,I0=100,R0=0,beta=0.001,mu=0.1,day=25)
p2 = simulate_SIR(S0=900,I0=100,R0=0,beta=0.002,mu=0.2,day=25)
p3 = simulate_SIR(S0=900,I0=100,R0=0,beta=0.004,mu=0.4,day=25)
plot = arrangeGrob(p1,p2,p3,ncol=1,top="Combined beta and mu effects on dynamics")
ggsave(filename="SIR3.png",plot=plot,width=5,height=10,path="Figures")
```



The infection and recovery rates are tremendously decisive for the system outcome. Proof of this is that the final states do not change when we alter the initial conditions but keep these parameters fixed.

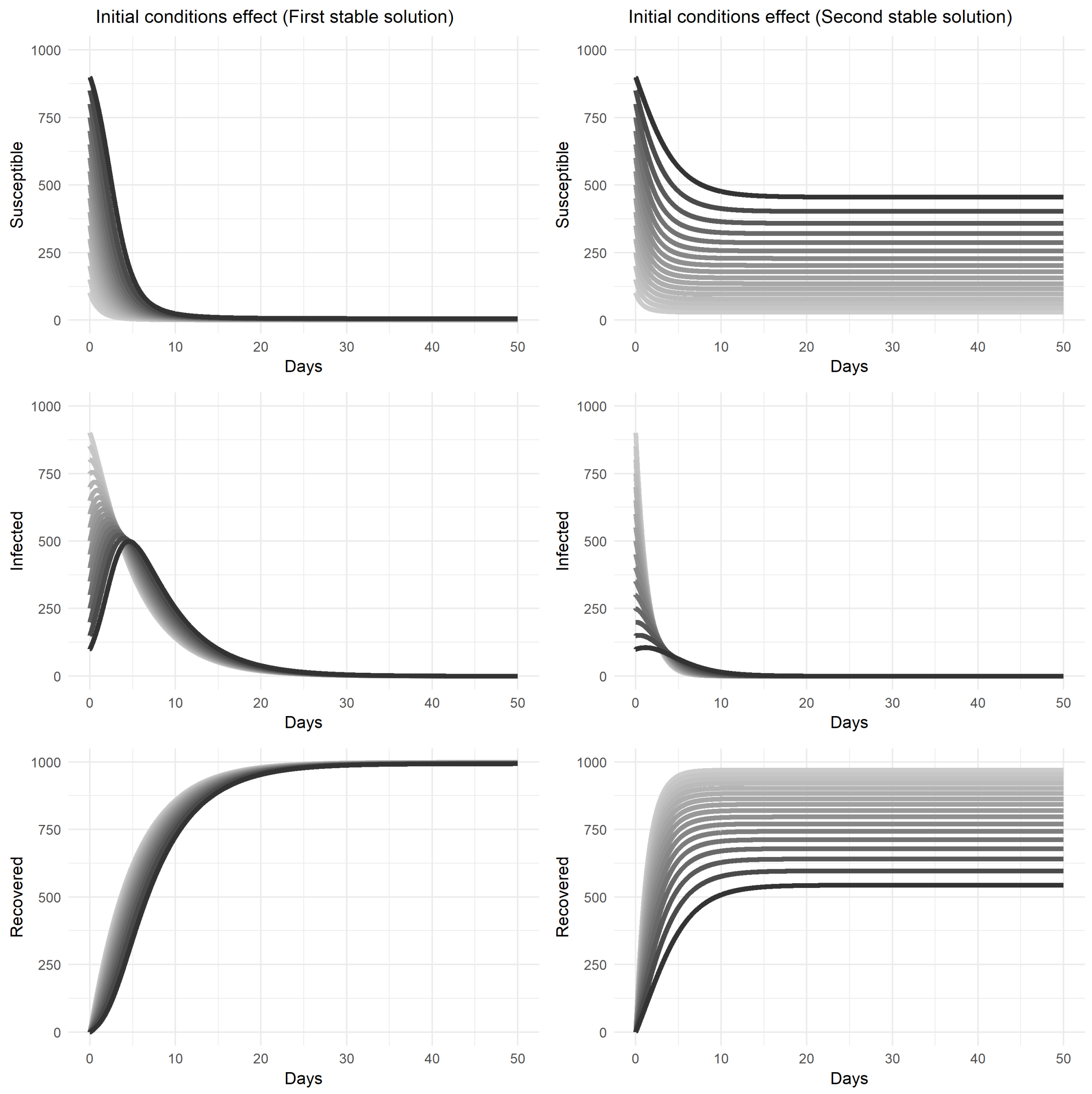
This is true for the first stable solution, in which all the individuals end up being recovered and no individual is susceptible or infected (S=0, I=0, R=N).

```
phase_space_df = data.frame()
for (i in seq(100,900,50))
{phase_space_df = rbind(phase_space_df,simulate_phase_space_SIR(i,1000-i,0,0.001,0.2,50))}

p1 = plot_phase_space_SIR(phase_space_df,"S","Susceptible","Conditions")
p2 = plot_phase_space_SIR(phase_space_df,"I","Infected","Conditions")
p3 = plot_phase_space_SIR(phase_space_df,"R","Recovered","Conditions")
plot1 = arrangeGrob(p1,p2,p3,ncol=1,top="Initial conditions effect (First stable solution)")
```

This is not completely true for the second stable solution, in which all the individuals are susceptible or recovered and no individual is infected (I=0, S+R=N). In such cases, the ratio between susceptible and recovered individuals can change, which makes sense because susceptible individuals can become recovered but the other way round is not possible in this model.

```
phase_space_df = data.frame()
for (i in seq(100,900,50))
{phase_space_df = rbind(phase_space_df,simulate_phase_space_SIR(i,1000-i,0,0.001,0.8,50))}

p1 = plot_phase_space_SIR(phase_space_df,"S","Susceptible","Conditions")
p2 = plot_phase_space_SIR(phase_space_df,"I","Infected","Conditions")
p3 = plot_phase_space_SIR(phase_space_df,"R","Recovered","Conditions")
plot2 = arrangeGrob(p1,p2,p3,ncol=1,top="Initial conditions effect (Second stable solution)")
plot = arrangeGrob(plot1,plot2,ncol=2)
ggsave(filename="SIR4.png",plot=plot,width=10,height=10,path="Figures")
```

This figure shows what was previously explained. Each line represents a different scenario.



We are interested in watching how the maximum incidence and his time of arrival evolve.

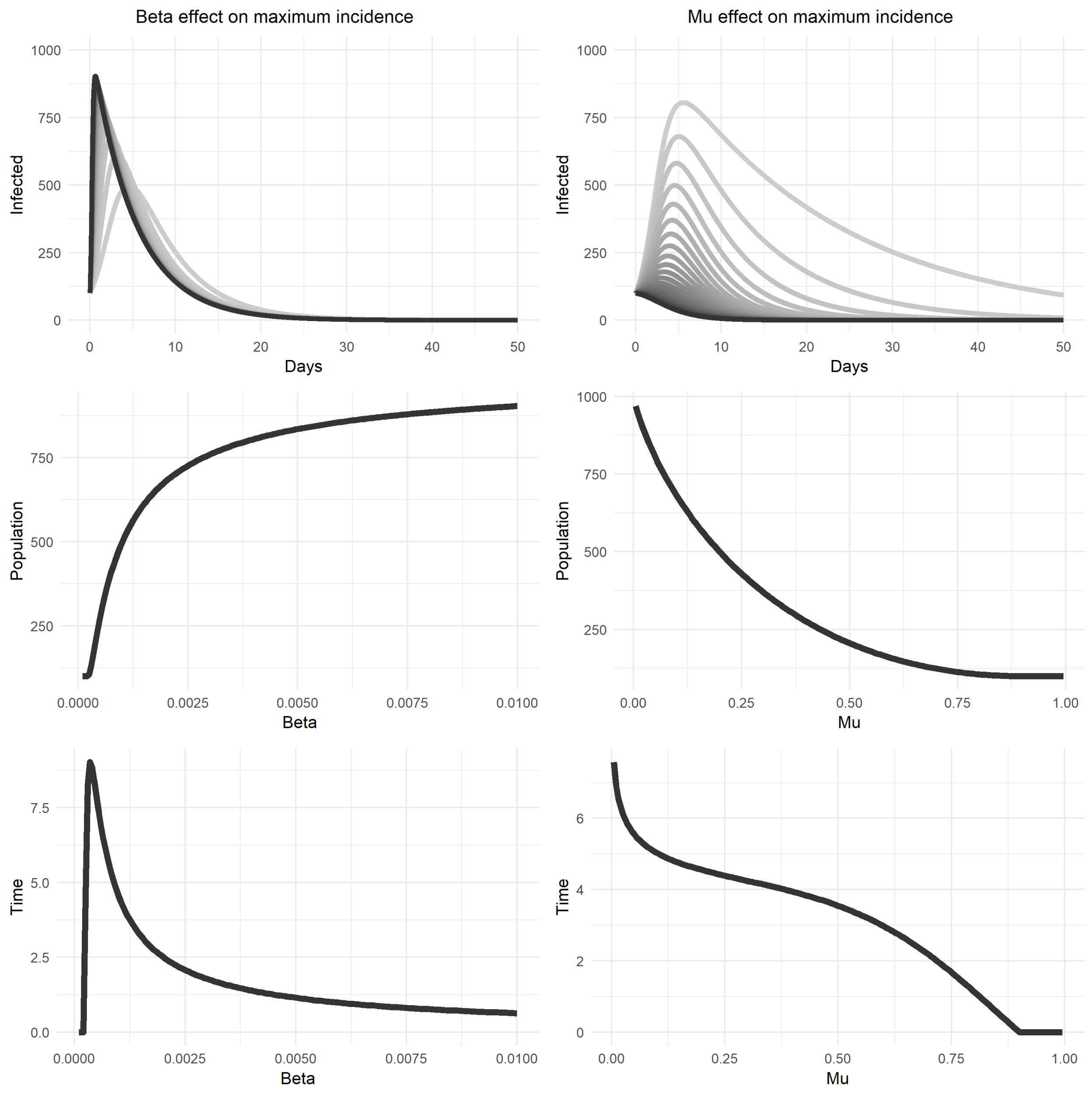
High values of β are associated with higher incidence. If we plot the maximum incidence for β, we see how the curve grows exponentially before reaching a plateau. If we plot the time at which that maximum incidence arrives, we can see how it starts in 0, grows fast and then decreases exponentially. The explanation is the following: when the infection is very slow, it cannot overtake the recovery, so the maximum incidence is the initial moment; when the infection is slow, it requires lot of time to overtake the recovery, so the maximum incidence appears late; when the infection is fast, it requires little time to overtake the recovery, so the maximum incidence appears early.

```r
phase_space_df = data.frame()
for (i in seq(0.001,0.01,0.0005))
{phase_space_df = rbind(phase_space_df, simulate_phase_space_SIR(900,100,0,i,0.2,50))}
Beta = c(); MI_population = c(); MI_time = c()
for (i in seq(0.0001,0.01,0.00005))
{
  df = simulate_phase_space_SIR(900,100,0,i,0.2,50)
  Beta = c(Beta,i)
  MI_population = c(MI_population,df[which.max(df$I),]$I)
  MI_time = c(MI_time,df[which.max(df$I),]$time)
}
evolution_df=data.frame(Beta=Beta,Population=MI_population,Time=MI_time)

p1 = plot_phase_space_SIR(phase_space_df,"I","Infected","Parameters")
p2 = plot_evolution(evolution_df,"Beta","Population")
p3 = plot_evolution(evolution_df,"Beta","Time")
plot1 = arrangeGrob(p1,p2,p3,ncol=1,top="Beta effect on maximum incidence")
```

Low values of μ are associated with higher incidence. If we plot the maximum incidence for μ, we see how the curve decreases exponentially before reaching a plateau. If we plot the time at which that maximum incidence arrives, we can see how it decreases slowly. The explanation is the following: when the recovery is slow, it allows for long infection periods; when the recovery is fast, it prevents from long infections periods.

```r
phase_space_df = data.frame()
for (i in seq(0.05,0.95,0.05))
{phase_space_df = rbind(phase_space_df,simulate_phase_space_SIR(900,100,0,0.001,i,50))}
Mu = c(); MI_population = c(); MI_time = c()
for (i in seq(0.005,0.995,0.005))
{
  df = simulate_phase_space_SIR(900,100,0,0.001,i,50)
  Mu = c(Mu,i)
  MI_population = c(MI_population,df[which.max(df$I),]$I)
  MI_time = c(MI_time,df[which.max(df$I),]$time)
}
evolution_df=data.frame(Mu=Mu,Population=MI_population,Time=MI_time)

p1 = plot_phase_space_SIR(phase_space_df,"I","Infected","Parameters")
p2 = plot_evolution(evolution_df,"Mu","Population")
p3 = plot_evolution(evolution_df,"Mu","Time")
plot2 = arrangeGrob(p1,p2,p3,ncol=1,top="Mu effect on maximum incidence")
plot = arrangeGrob(plot1,plot2,ncol=2)
ggsave(filename="SIR5.png",plot=plot,width=10,height=10,path="Figures")
```

This figure shows what was previously explained. For top plots, each line represents a different scenario.



4 Extended biological processes: The role of space

Cellular automata (CA) are an idealization of physical systems composed of a discrete lattice of homogeneous units, the cells, that can have a finite set of states. The cells evolve in parallel at discrete time steps; such evolution, which is guided by a deterministic transition function, only takes into account the state of cells in the local neighborhood. The idea behind CA is that very simple rules can lead to very complex patterns. In this way, even perfect knowledge at a local level might not allow us to predict at a global level.

Why are CA useful? Although it is common in research to isolate the different components of a system and study them separately, the real world is made of interacting processes. The rules that describe CA behavior, even though simple, are capable of replicating such interactions. In other word, CA models can actually be realistic, which makes them useful for simulating scenarios. This idea of simplicity is usually maintained for the number of dimensions (most applied problems can be solved by using lattices of no more than 3 dimensions) and the number of states. Were it different, it would be difficult to get any insight.

For a one-dimensional lattice CA of two states, each cell's neighborhood is composed of the two nearest cells. In such scenario, there are $2^3 = 8$ possible configurations for a cell and its neighbors. The transition rule defining the CA must specify the resulting state for each of these possibilities, which makes up to a total of $2^8 = 256$ possible rules. Stephen Wolfram proposed a scheme, known as the Wolfram Code, to assign each rule a number from 0 to 255. To know the behavior of each rule, we just have to convert the decimal number into binary number and consider each digit as the resulting state of the 8 possible configurations, ordered in a binary way.

Starting with random initial conditions, Wolfram observed the behavior of each rule in many simulations. He was then able to classify the rules in four classes: Class 1 rules produce homogeneity in all cell states; Class 2 rules produce simple regular patterns; Class 3 rules produce random-looking configurations with some regular patterns; Class 4 rules produce complex regular patterns that propagate through the lattice.

As we increase either the number of dimensions or the number of states, more configurations and rules appear. For two-dimensional lattice, the most well CA is probably John Conway's "Game of Life", which turned out to be a universal Turing machine, capable of carrying out any algortithmic procedure given a suitable selection of initial conditions.

4.1 Study the dynamics for a set of transition rules

In this exercise, we will study the dynamics for a set of transition rules in a one-dimensional lattice CA of two states. We should then be able to classify these rules in one of the existing Wolfram classes.

This function gives an output for each transition rule and configuration.

```r
#It receives a transition rule and a configuration
#It converts the rule in decimal number into a binary number of 8 digits
#It returns an output for each neighborhood configuration

solve_configuration = function(rule_decimal,left,center,right)
{
  rule_binary = strsplit(intToBin(rule_decimal),"")[[1]]
  while (length(rule_binary)<8) {rule_binary=c(0,rule_binary)}
  if (rule_binary[1]==1 & left==1 & center==1 & right==1) return(1)
  if (rule_binary[2]==1 & left==1 & center==1 & right==0) return(1)
  if (rule_binary[3]==1 & left==1 & center==0 & right==1) return(1)
  if (rule_binary[4]==1 & left==1 & center==0 & right==0) return(1)
  if (rule_binary[5]==1 & left==0 & center==1 & right==1) return(1)
  if (rule_binary[6]==1 & left==0 & center==1 & right==0) return(1)
  if (rule_binary[7]==1 & left==0 & center==0 & right==1) return(1)
  if (rule_binary[8]==1 & left==0 & center==0 & right==0) return(1)
  return(0)
}
```

This function creates a matrix representing the evolution of the CA.

```r
#It receives a transition rule, initial condition and number of iterations
#It creates a matrix and assigns the initial conditions to the first row
#It iterates over each cell
#It retrieves the neighborhood configuration and includes the corresponding output
#It returns a matrix representing the evolution from the initial conditions

create_matrix = function(rule_decimal,initial_condition,number_iterations)
{
  number_columns = length(initial_condition)
  number_rows = number_iterations
  matrix = matrix(data=NA,ncol=number_columns,nrow=number_rows)
  matrix[1,] = initial_condition
  for (row in 2:number_rows)
  {
    for (column in 1:number_columns)
    {
      center = matrix[row-1,column]
      if (column==1) {left=matrix[row-1,number_columns]}
      else {left=matrix[row-1, column-1]}
      if (column==number_columns) {right=matrix[row-1,1]}
      else {right=matrix[row-1,column+1]}
      matrix[row,column] = solve_configuration(rule_decimal, left, center, right)
    }
  }
  return(matrix)
}
```

This function plots a given matrix.

```r
plot_matrix = function(matrix,title)
{
  melted_matrix = melt(t(matrix))
  melted_matrix = melted_matrix[nrow(melted_matrix):1,]
  class(melted_matrix$value) = "character"
  ggplot(melted_matrix, aes(x=Var1,y=sort(Var2,decreasing=FALSE),fill=value)) +
  geom_tile() + coord_equal() +
  scale_fill_manual(values=c("0"="white", "1"="black")) +
  scale_x_discrete(expand=c(0,0)) +
  scale_y_discrete(expand=c(0,0)) +
  ggtitle(title) +
  theme_void() +
  theme(legend.position="none") +
  theme(plot.title=element_text(hjust=0.5,size=10))
}
```

We define the initial conditions that are going to be considered.

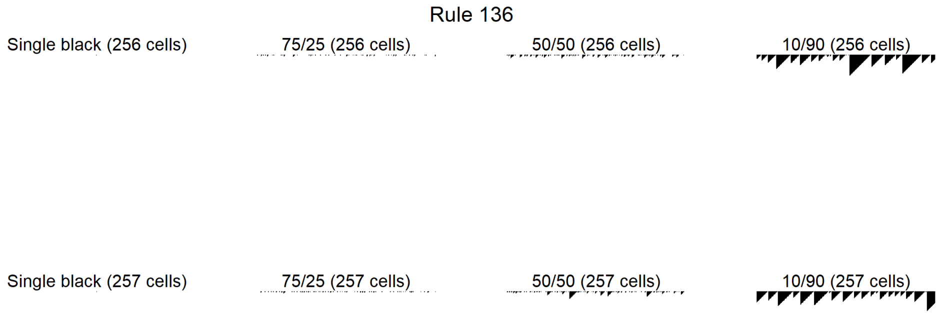
```r
#All white cells (0) with one black cell (1) in the middle
init_cond_AM_256 = c(rep(0,127),1,rep(0,128))
init_cond_AM_257 = c(rep(0,128),1,rep(0,128))

#75% white cells (0) and 25% black cells (1) at randomly chosen positions
init_cond_7525_256 = rbinom(256,1,0.25)
init_cond_7525_257 = rbinom(257,1,0.25)

#Half white cells (0) and half black cells (1) at randomly chosen positions
init_cond_5050_256 = rbinom(256,1,0.5)
init_cond_5050_257 = rbinom(257,1,0.5)

#10% white cells (0) and 90% black cells (1) at randomly chosen positions
init_cond_1090_256 = rbinom(256,1,0.9)
init_cond_1090_257 = rbinom(257,1,0.9)
```

We run the simulation for the chosen rules (136,48,45,137).

```r
rules=c(136,48,45,137)
for (rule in rules)
{
  p1 = plot_matrix(create_matrix(rule,init_cond_AM_256,300),"Single black (256 cells)")
  p2 = plot_matrix(create_matrix(rule,init_cond_7525_256,300),"75/25 (256 cells)")
  p3 = plot_matrix(create_matrix(rule,init_cond_5050_256,300),"50/50 (256 cells)")
  p4 = plot_matrix(create_matrix(rule,init_cond_1090_256,300),"10/90 (256 cells)")
  p5 = plot_matrix(create_matrix(rule,init_cond_AM_257,300),"Single black (257 cells)")
  p6 = plot_matrix(create_matrix(rule,init_cond_7525_257,300),"75/25 (257 cells)")
  p7 = plot_matrix(create_matrix(rule,init_cond_5050_257,300),"50/50 (257 cells)")
  p8 = plot_matrix(create_matrix(rule,init_cond_1090_257,300),"10/90 (257 cells)")
  plot = arrangeGrob(p1,p2,p3,p4,p5,p6,p7,p8,ncol=4,top=paste0("Rule ",rule))
  ggsave(filename=paste0("Rule",rule,".png"),plot=plot,width=8,height=4,path="Figures")
}
```

The rule 136 (10001000) has the following dynamics:

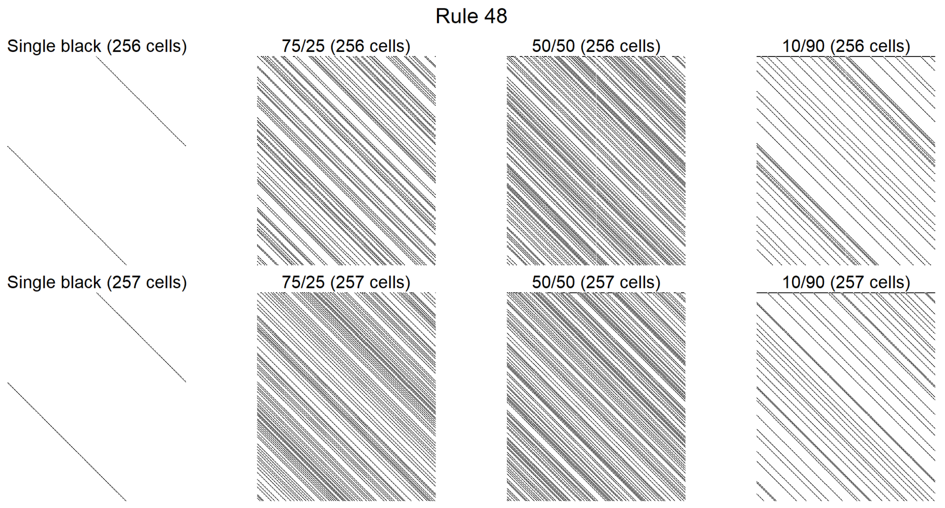
Neighborhood configuration	111	110	101	100	011	010	001	000
Middle cell resulting state	1	0	0	0	1	0	0	0



It is a clear example of Class 1, as it quickly produces homogeneity in all cell states. The fact that black cells appear only from X-B-B explains the following phenomena: the presence of any white cell turns everything white rapidly; the greater the amount of black cells, the latest this homogeneity arrives; the regular patterns that appear until this homogeneity arrive are right-sided triangles.

The rule 48 (00110000) has the following dynamics:

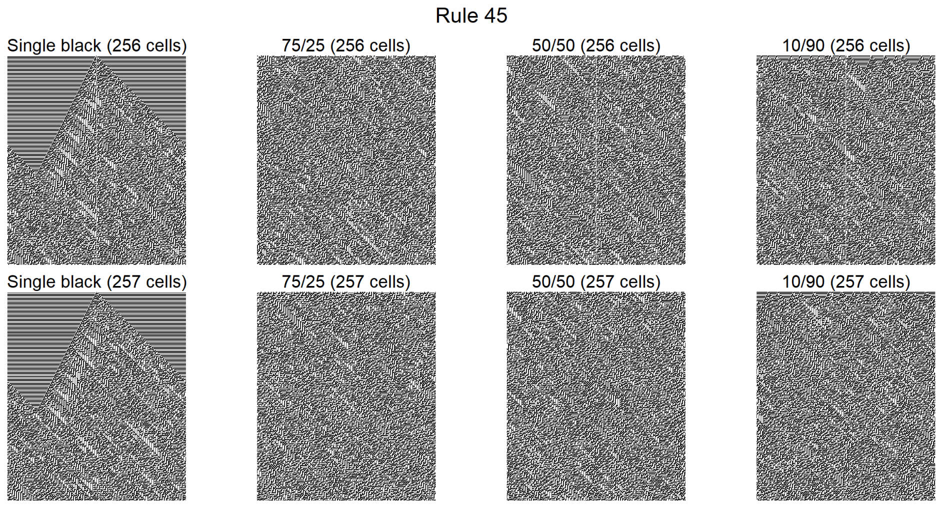
Neighborhood configuration	111	110	101	100	011	010	001	000
Middle cell resulting state	0	0	1	1	0	0	0	0



It is a clear example of Class 2, as it produces simple regular patterns (diagonal lines). The fact that black cells appear only from B-W-? explains the following phenomena: the greater the parity, the greater the number of lines; the lines are directed towards the right edge; there cannot be two black lines together since a white space between them is required.

The rule 45 (00101101) has the following dynamics:

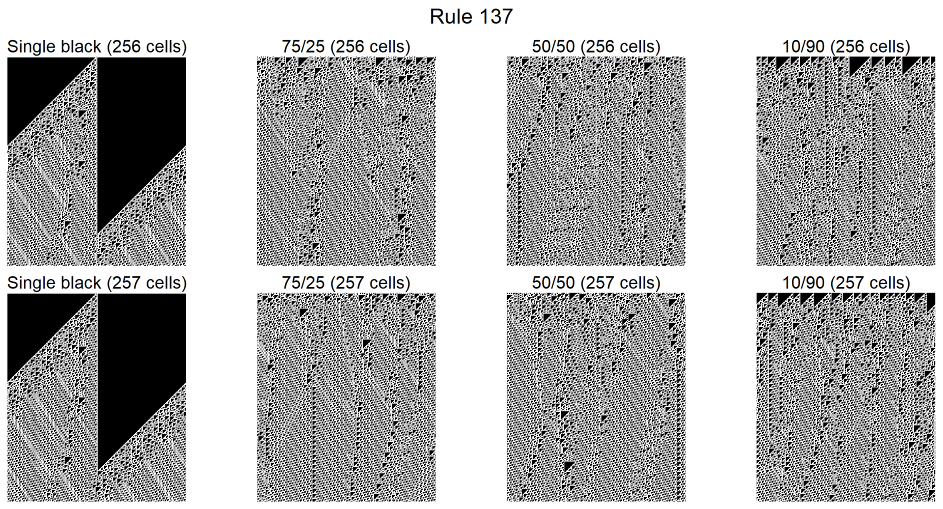
Neighborhood configuration	111	110	101	100	011	010	001	000
Middle cell resulting state	0	0	1	0	1	1	0	1



It is a clear example of Class 3, as it produces random-looking configurations with some regular patterns (horizontal lines). The fact that black cells appear from W-W-W and that white cells appear from B-B-B explains the following phenomena: regular patterns appear from large arrays of cells with the same state; the greater the parity, the sooner random-looking patterns become predominant.

The rule 137 (10001001) has the following dynamics:

Neighborhood configuration	111	110	101	100	011	010	001	000
Middle cell resulting state	1	0	0	0	1	0	0	1



It is a clear example of Class 4, as it produces complex regular patterns (triangles) that propagate through the lattice. The fact that black cells appear from ?-B-B and W-W-W explains the following phenomena: the regular patterns that appear are right sided triangles; no homogeneity of white cells arrives.

4.2 Design a cellular automaton

In these days where political confrontation is so common, I thought it would be interesting to design a CA that would reproduce an scenario of political ideology diversity. In this CA in particular, I have included 3 different political ideologies (left-wing, center and right-wing). The transitions between such states depend on two things: the political ideology of the individual (the personal convictions) and the predominant political ideology of its neighborhood (the collective pressure).

This function is the core of the CA, changing a bidimensional matrix (which represents the world of the CA) according to the transition rules. These are based on the logical idea that left-wings are closer to centers than to right-wings, and vice versa. Thus, the collective pressure needed will be different.

```
#It receives the input matrix and the matrix size
#It iterates over each cell
#It defines the upper row, lower row, left column and right column
#This is done taking boundary conditions into consideration
#It determines the neighborhood
#It determines the differences between left-wing and right-wing
#If a cell is left-wing (0), it will...
#Change to center if there are 1 or 2 more right-wing
#Change to right-wing if there are at least 3 more right-wing
#If a cell is center (1), it will...
#Change to left-wing if there is at least 2 more left-wing
#Change to right-wing if there is at least 2 more right-wing
#If a cell is right-wing (2), it will...
#Change to center if there are 1 or 2 more left-wing
#Change to left-wing if there are at least 3 more left-wing

change_matrix = function(input_matrix,matrix_size)
{
  output_matrix = matrix(NA,nrow=matrix_size,ncol=matrix_size)
  for (row in 1:matrix_size)
  {
    for (column in 1:matrix_size)
    {

      if (row==1)
      {
        upper_row=matrix_size; lower_row=row+1
        if (column==1){left_column=matrix_size; right_column=column+1}
        else if (column==matrix_size){left_column=column-1; right_column=1}
        else {left_column=column-1; right_column=column+1}
      }
      else if (row==matrix_size)
      {
        upper_row=row-1; lower_row=1
        if (column==1){left_column=matrix_size; right_column=column+1}
        else if (column==matrix_size){left_column=column-1; right_column=1}
        else {left_column=column-1; right_column=column+1}
      }
      else if (column==1)
      {
        upper_row=row-1; lower_row=row+1
        left_column=matrix_size; right_column=column+1
      }
```

```r
      else if (column==matrix_size)
      {
        upper_row=row-1; lower_row=row+1
        left_column=column-1; right_column=1
      }
      else
      {
        upper_row=row-1; lower_row=row+1
        left_column=column-1; right_column=column+1
      }

      upper = input_matrix[upper_row,column]
      lower = input_matrix[lower_row,column]
      left = input_matrix[row,left_column]
      right = input_matrix[row,right_column]
      upper_left = input_matrix[upper_row,left_column]
      upper_right = input_matrix[upper_row,right_column]
      lower_left = input_matrix[lower_row,left_column]
      lower_right = input_matrix[lower_row,right_column]
      neighborhood = c(upper_left,upper,upper_right,left,right,lower_left,lower,lower_right)
      number_left_wing = sum(neighborhood==0)
      number_right_wing = sum(neighborhood==2)
      difference = number_right_wing - number_left_wing

       if (input_matrix[row,column]==0)
       {
         if (difference %>% between(-8,0)){output_matrix[row,column]=0}
         else if (difference %>% between(1,2)){output_matrix[row,column]=1}
         else if (difference %>% between(3,8)){output_matrix[row,column]=2}
       }

       else if(input_matrix[row,column]==1)
       {
         if (difference %>% between(-8,-2)){output_matrix[row,column]=0}
         else if (difference %>% between(-1,1)){output_matrix[row,column]=1}
         else if (difference %>% between(2,8)){output_matrix[row,column]=2}
       }

      if (input_matrix[row,column]==2)
      {
        if (difference %>% between(-8,-3)){output_matrix[row,column]=0}
        else if (difference %>% between(-2,-1)){output_matrix[row,column]=1}
        else if (difference %>% between(0,8)){output_matrix[row,column]=2}
      }
    }
  }
  return(output_matrix)
}
```

This function generates a list of matrix representing the evolution the CA.

```r
create_matrix_list = function(number_iterations,matrix_size)
{
  initial_matrix = matrix(rdunif(matrix_size^2,0,2),nrow=matrix_size,ncol=matrix_size)
  matrix_list = list(initial_matrix)
  for (iteration in 2:number_iterations)
  {matrix_list[[iteration]]=change_matrix(matrix_list[[iteration-1]],matrix_size)}
  return(matrix_list)
}
```

This function plots a given matrix.

```r
plot_matrix = function(matrix,title)
{
  melted_matrix = melt(t(matrix))
  class(melted_matrix$value) = "character"
  plot = ggplot(melted_matrix,aes(x=Var1,y=Var2,fill=value)) +
  geom_tile() +
  coord_equal() +
  scale_fill_manual(values=c("0"="red", "1"="white","2"="blue")) +
  scale_x_discrete(expand=c(0,0)) +
  scale_y_discrete(expand=c(0,0)) +
  ggtitle(title) +
  theme_void() +
  theme(legend.position="none") +
  theme(plot.title=element_text(hjust=0.5,size=10))
  return(plot)
}
```

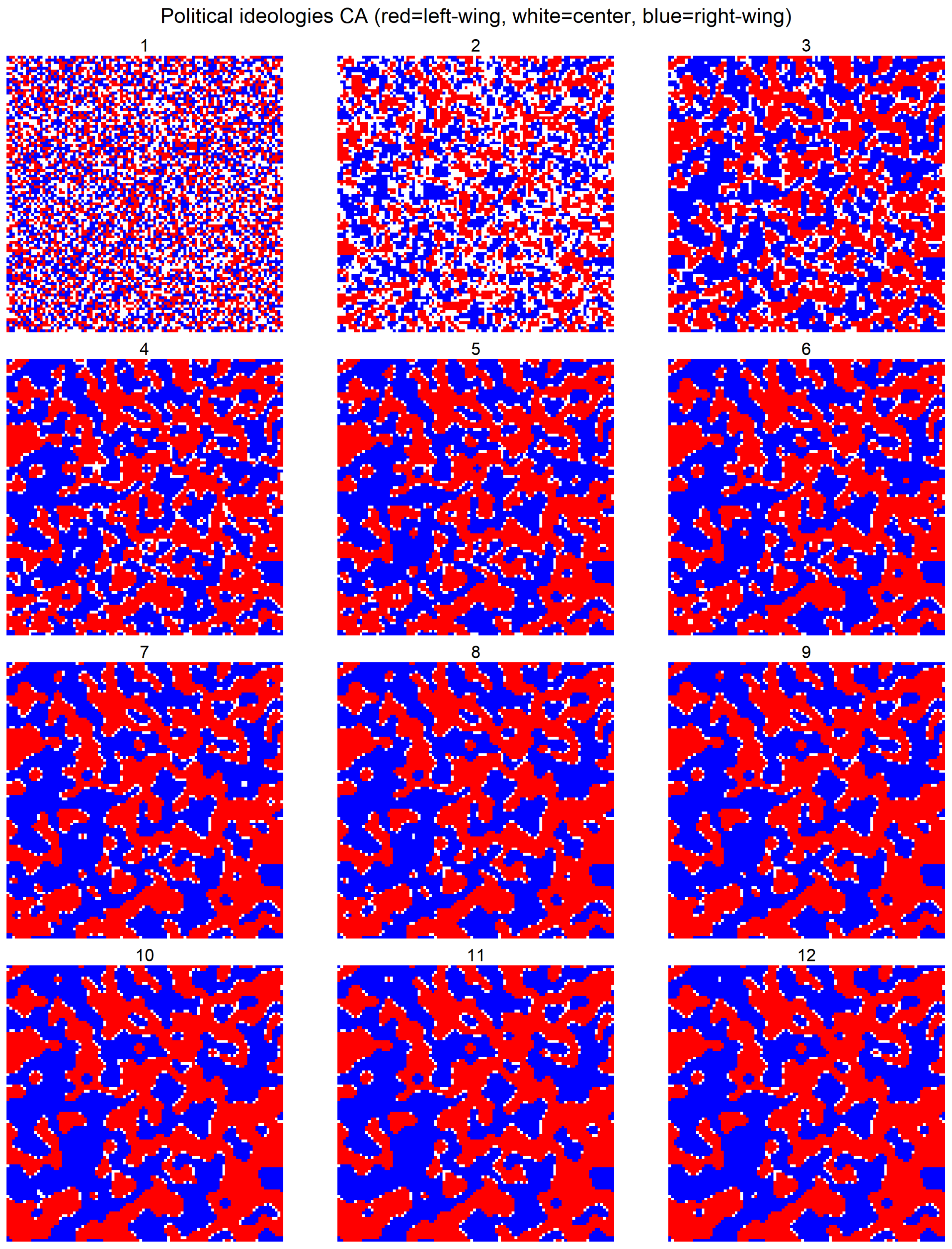
We run the simulation.

```r
number_iterations=12
matrix_size=100
matrix_list = create_matrix_list(number_iterations,matrix_size)
```

We obtain the plots for the simulation.

```r
for (iteration in 1:number_iterations)
{assign(paste0("CA",iteration),plot_matrix(matrix_list[[iteration]],iteration))}
plot = arrangeGrob(CA1,CA2,CA3,CA4,CA5,CA6,CA7,CA8,CA9,CA10,CA11,CA12,ncol=3,
top="Political ideologies CA (red=left-wing, white=center, blue=right-wing)")
ggsave(filename="CA.png",plot=plot,width=8,height=10,path="Figures")
```

The results are the followings:



The CA behaves in a way that left-wing individuals are segregated from right-wing individuals, being both in similar proportions. In addition, center individuals quickly loss importance and stay in between left-wing and right-wing clusters. Both these patterns arise quickly from a completely random distribution and stay stable from then on.

The CA exhibits, therefore, behaviors that appear in real life: the segregation in space between different political ideologies (certain regions have a clear political tendency while others have the opposite) and the disappearance of center political ideologies due to indecision.

A proper study of this problem would require, however, from considering different transition rules (we used logical reasoning to establish them but a field study might proof others to be more reliable) and perturbation events (the real world is full of event that affect people's political ideology).

5 Extended biological processes: The role of networks

When facing an interacting system, some people study the nature of the individual components while others study the nature of their connections. A third aspect that is crucial to the behavior of these systems is the pattern of connections between components, which can be represented as a network.

A network is, in its simplest form, a collection of vertices joined in pairs by edges. In order to capture more detail about a specific system, vertices and edges can be labelled with additional information. Even so, a lot of information is lost in the process of reducing a full system to its network representation.

Networks can be classified according to their edges characteristics (they can be single or multiple, weighted or not weighted, directed or not directed. . .). As for the properties, we should talk about centrality, assortativity and modularity.

Centrality quantifies how important vertices are in a network. There is a wide variety of mathematical measures of vertex centrality that focus on different concepts and definitions of what it means to be central in a network. The degree centrality considers the number of edges attached to each vertex. The eigenvector centrality suggests that not all the vertices are equivalent, so the importance is increased by having connections to vertices that are themselves important. The closeness centrality measures the mean distance from one vertex to the others, assuming that this will reflect their capacity to access information. The betweenness centrality measures the extent to which a vertex lies on paths between others, assuming that this will reflect their influence over the flow of information. One of the main applications of centrality quantification is the identification of hubs, these are, vertices with outstanding importance. Hubs, despite few in number, play a key role in the performance and behavior of the network, especially when it comes to transportation phenomena.

Assortativity measures the fraction of edges in a network that connect vertices of the same type. It is calculated in relation to the fraction of these type of edges that would be find in a randomly positioned network. This property gives us information about whether the network is organized by some characteristic.

The modularity measures the strength of division of a network into modules (groups, clusters, communities).

5.1 Study the effects of heterogeneity in the number of contacts

In this exercise, we will perform a numerical simulation of a rumor propagation process that starts with a single individual. We build networks of 100 individuals with the models: Randomly Connected, Small World and Preferential Attachment. For each model, we study the rumor propagation in a single simulation, as well as some patterns common to several simulations.

This function counts the number of individuals to which the rumor can get.

```r
#It receives a network in its first iteration
#It returns the number of individuals connected to the only one who knows the rumor

rumor_range = function(network)
{
  know = which(V(network)$color=="red")
  cluster_know = components(network)$membership[know]
  connected_vertices = sum(components(network)$membership==cluster_know)
  return(connected_vertices)
}
```

This function spreads the rumor across a network.

```r
#It receives a network
#It determines which individuals know the rumor
#It expands the rumor to their immediate neighbors
#It returns the updated network

spread_rumor = function(network)
{
  know = which(V(network)$color=="red")
  will_know = c()
  for (vertix in know)
  {
    connected_to_know = which(network[vertix]==1)
    will_know = c(will_know,connected_to_know)
  }
  V(network)$color[will_know] = "red"
  return(network)
}
```

This function simulates a complete rumor propagation for a given network model.

```r
#It receives the model of network that wants to be built
#It creates the network and fixes the positions
#It starts the first iteration by setting the rumor in one random vertex
#It spreads the rumor until it reaches all possible vertices
#Note that, depending in the model, the rumor cannot always reach all vertices
#It returns a list with information on the rumor evolution

network_simulation = function(type,plot)
{

  if (type=="RC"){network=erdos.renyi.game(n=100,p=0.05,type="gnp")}
  if (type=="SW"){network=sample_smallworld(dim=1,size=100,nei=4,p=0.05)}
  if (type=="PA"){network=sample_pa(n=100,power=1,directed=FALSE)}
  positions = layout_with_fr(network)

  iteration = 1
  V(network)$color = "lightblue"
  V(network)$color[sample(1:100,1)] = "red"
  if(plot==TRUE){plot(network,layout=positions,vertex.size=10,vertex.label=NA)}
  iterations = c(1)
  real = c(1)
  theory = c(1)

  rumor_range = rumor_range(network)
  while (sum(V(network)$color=="red")<rumor_range)
  {
    iteration = iteration+1
    network = spread_rumor(network)
    if(plot==TRUE){plot(network,layout=positions,vertex.size=10,vertex.label=NA)}
    iterations = c(iterations,iteration)
    real = c(real,length(which(V(network)$color=="red")))
    theory = c(theory,theory[iteration-1]+theory[iteration-1]*mean(degree(network)))
  }

  simulation = list()
  simulation[["Degrees"]] = degree(network)
  simulation[["Iterations"]] = iterations
  simulation[["Real"]] = real
  simulation[["Theory"]] = theory
  return(simulation)
}
```

This function plots the rumor evolution.

```r
plot_evolution = function(iterations,real,theory,tags)
{
  evolution = data.frame(Iterations=iterations,Real=real,Theory=theory)
  evolution = gather(evolution,"Real","Theory",key="Evolution",value="Number")
  plot = ggplot(evolution,aes(x=Iterations,y=Number,color=Evolution)) +
  geom_line(size=2) +
  scale_y_log10(breaks=trans_breaks("log10",function(x)10^x),
  labels=trans_format("log10",math_format(10^.x))) +
  scale_x_continuous(breaks=iterations) +
  ylab("Numbers of informed individuals") +
  xlab("Number of generations") +
  ggtitle(tags) +
  theme_minimal() +
  theme(legend.position=c(0.15,0.85)) +
  theme(plot.title=element_text(hjust=0.5))
  return(plot)
}
```

This function plots the distribution of a vector of values.

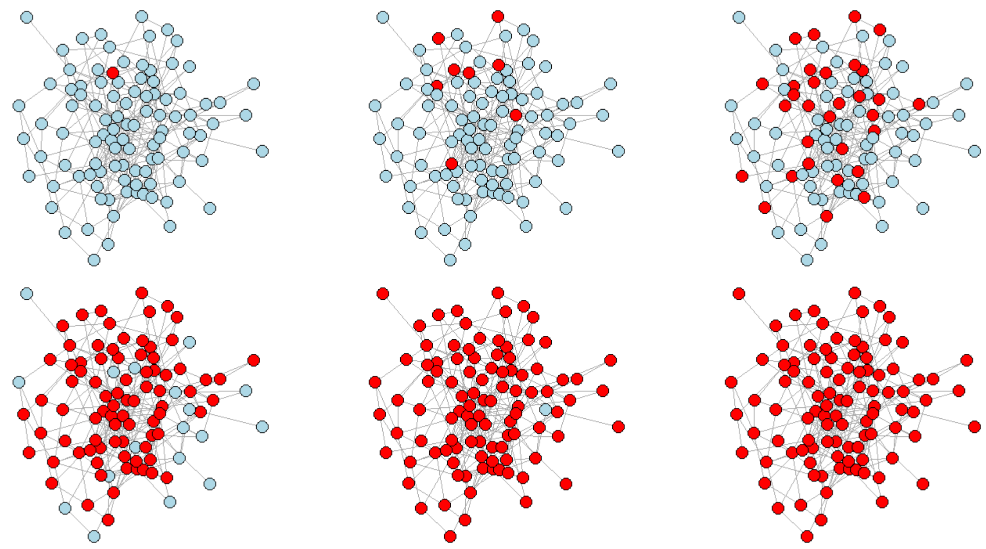
```r
plot_distribution = function(vector,tags)
{
  different_values = as.integer(names(table(vector)))
  vector_df = data.frame(values=vector)
  plot = ggplot(vector_df,aes(x=values)) +
  geom_histogram(binwidth=1) +
  scale_x_continuous(breaks=different_values) +
  xlab(tags[1]) +
  ylab(tags[2]) +
  ggtitle(tags[3]) +
  theme_minimal() +
  theme(plot.title=element_text(hjust=0.5))
  return(plot)
}
```

A random graph is a network in which some specific parameters take fixed values but the network is random in other respects. Once simple example of random graph is the Erdos-Renyi model, which fixes the number of vertices and edges. One slightly different model fixes the number of vertices and then places an edge between each distinct pair of vertices with a given probability. We will refer to this model as Randomly Connected.

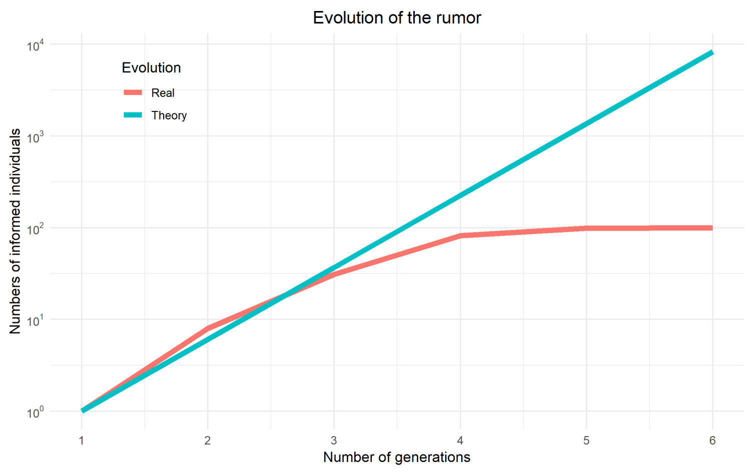
We do the analysis for the Randomly Connected model.

```r
par(mar=c(0,0,0,0))
par(mfrow=c(3,2))
set.seed(2015)
RC_one_simulation = network_simulation("RC",TRUE)

RC_iterations = c(); RC_degrees = c()
for (simulation in 1:500)
{
  RC_simulation = network_simulation("RC",FALSE)
  if (tail(RC_simulation$Real,n=1)>80)
  {RC_iterations = c(RC_iterations,length(RC_simulation$Iterations))}
  RC_degrees = c(RC_degrees,RC_simulation$Degrees)
}

p1 = plot_evolution(RC_one_simulation$Iterations,RC_one_simulation$Real,
RC_one_simulation$Theory,"Evolution of the rumor")
p2 = plot_distribution(RC_iterations,c("Number of generations",
"Number of simulations","Number of generations for rumor transmission"))
p3 = plot_distribution(RC_degrees,c("Degrees","Number of vertices","Degree distribution"))
plot = arrangeGrob(p2,p3,ncol=2)
ggsave(filename="NW2.png",plot=p1,width=8,height=5,path="Figures")
ggsave(filename="NW3.png",plot=plot,width=12,height=6,path="Figures")
```

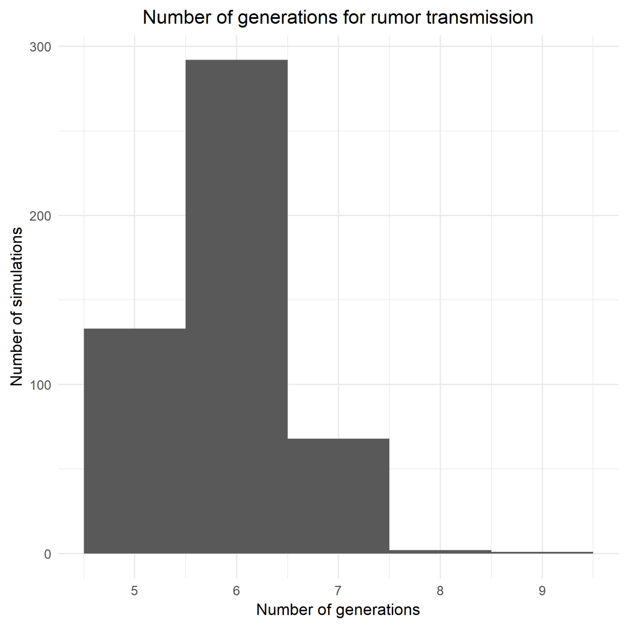
This figure shows the rumor propagation for a Randomly Connected model simulation.

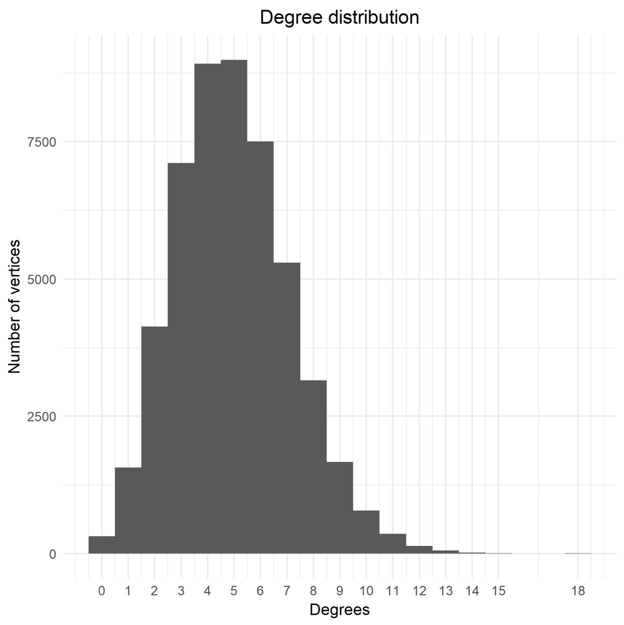


This figure shows the evolution in time for such rumor propagation.



This figure includes some metrics from 500 Randomly Connected model simulations.





The transmission of the rumor is usually complete (all the individuals receive the rumor) but can also be incomplete (not all the individuals receive the rumor). The explanation to this is that, when the connection between vertices depends on a probability, some vertex might not be connected. If the rumor starts in one vertex belonging to a big connected structure (>80), the rumor propagation is considered as valid for the duration analysis. If the rumor starts in one vertex belonging to a small connected structure (<80), the rumor propagation is considered invalid for the duration analysis. The transmission of the rumor is generally fast, with a mean of 6 generations, and moderately variability (despite randomness, the fact that the number of connections is stable results in networks behaving in a similar way). The degree distribution resembles a normal distribution, centered around 5, where central degrees do not gather most of the values. This is coherent with the process of building the edges.

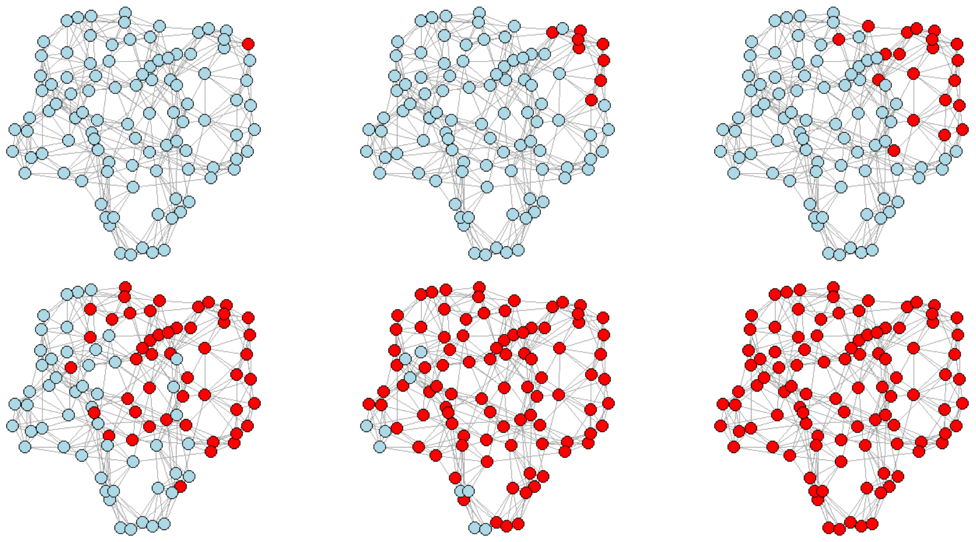
It is quite obvious that this results depend a lot on the parameters we have chosen. We used 0.05 as the probability of two random vertex being connected; increasing such probability would allow for a faster and complete rumor propagation.

Random models are capable of capturing the short path length feature of real world networks (the path distance between most vertices in a network is small). Circular models, where vertices are arranged in a one-dimensional circle and connected to their neighbors, is capable of capturing the transitivity feature of real world networks (two neighbors of a vertex will probably be neighbors themselves). None of them, however, is capable of capturing both. The Watts-Strogatz model displays both transitivity and short path lengths simultaneously. It interpolates between both models, rewiring edges from the circle to random positions and creating shortcuts. The structure of the network depends, thus, from the rewiring probability, which allows going from a completely regular to a completely random network. We will refer this model as Small World.

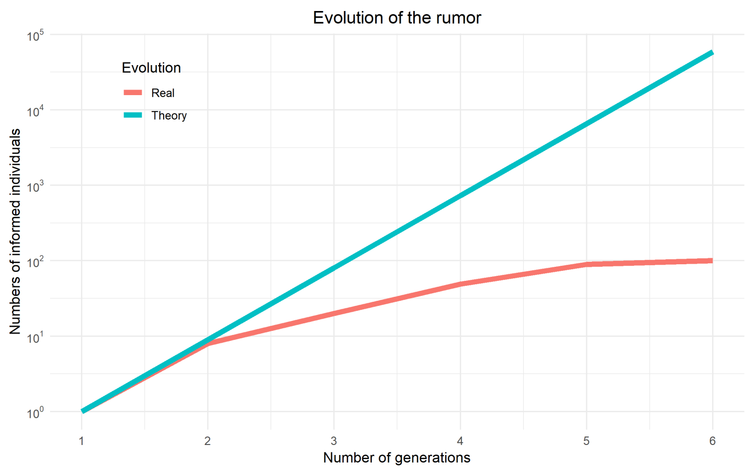
We do the analysis for the Small World model.

```r
par(mar=c(0,0,0,0))
par(mfrow=c(3,2))
set.seed(2020)
SW_one_simulation = network_simulation("SW",TRUE)

SW_iterations = c(); SW_degrees = c()
for (simulation in 1:500)
{
  SW_simulation = network_simulation("SW",FALSE)
  SW_iterations = c(SW_iterations,length(SW_simulation$Iterations))
  SW_degrees = c(SW_degrees,SW_simulation$Degrees)
}

p1 = plot_evolution(SW_one_simulation$Iterations,SW_one_simulation$Real,
SW_one_simulation$Theory,"Evolution of the rumor")
p2 = plot_distribution(SW_iterations,c("Number of generations",
"Number of simulations", "Number of generations for rumor transmission"))
p3 = plot_distribution(SW_degrees,c("Degrees","Number of vertices","Degree distribution"))
plot = arrangeGrob(p2,p3,ncol=2)
ggsave(filename="NW5.png",plot=p1,width=8,height=5,path="Figures")
ggsave(filename="NW6.png",plot=plot,width=12,height=6,path="Figures")
```

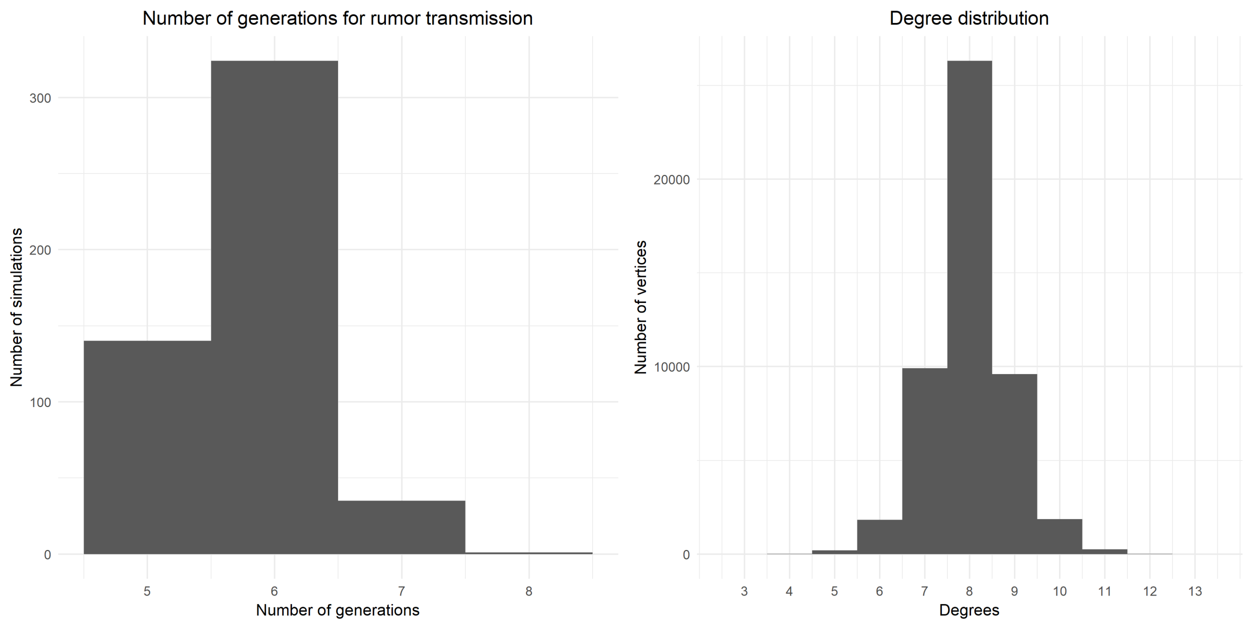
This figure shows the rumor propagation of a Small World model simulation.



This figure shows the evolution in time for such rumor propagation.



This figure includes some metrics from 500 Small World model simulations.



The transmission of the rumor is complete (all the individuals receive the rumor). The explanation to this is that all vertices are connected to the network so the rumor can reach them all. The transmission of the rumor is generally fast, with a mean of 6 generations, and moderately variability (although some randomness is introduced, networks follow the same general structure: most vertices connect neighbors but some connect remote vertices). The degree distribution resembles a normal distribution, centered around 8, where central degrees do gather most of the values. These is coherent with the process of building the edges as we are transitioning from a network where all vertices have the same degree to a network where such degree depends on randomness.

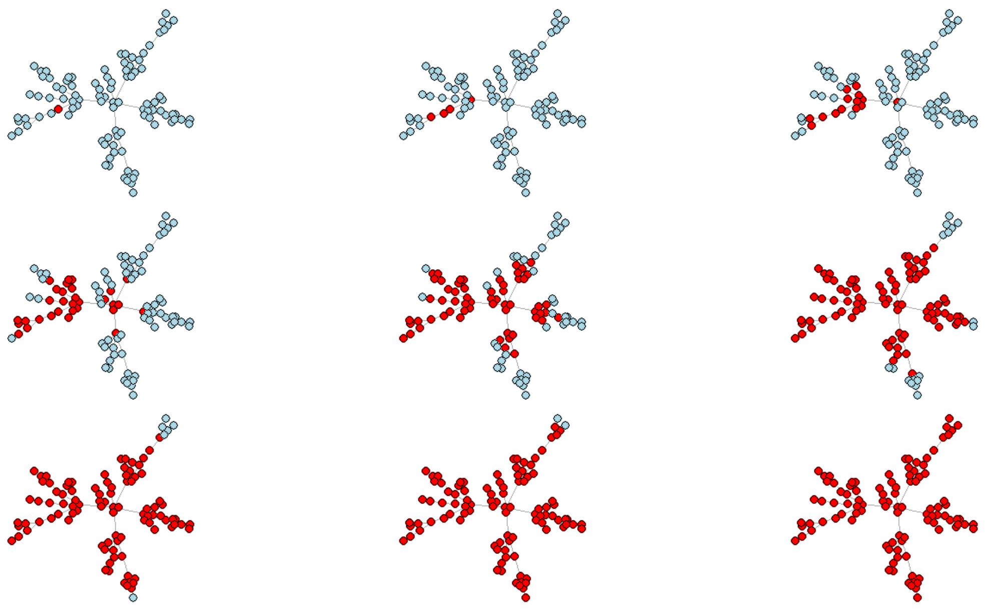
It is quite obvious that the results we obtained depend a lot on the parameters we have chosen. We used 4 as the number of vertices to which each vertex is connected; increasing such quantity would allow for a faster rumor propagation. We also used 0.05 as the rewiring probability; increasing such probability would make the networks more alike random graphs.

Many real-world networks are observed to have a degree distribution that follows power laws. The Barabasi-Albert model, which is based on Price previous ideas, is capable of retrieving such a distribution. The key feature of this model is that connections are made to vertices with a probability proportional to their current degree, which results in some vertices being clearly more important than others. We will refer to this model as Preferential Attachment.

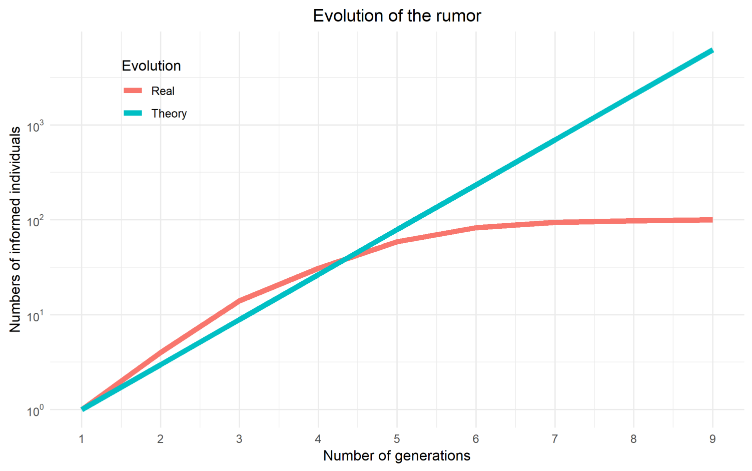
We do the analysis for the Preferential Attachment model.

```r
par(mar=c(0,0,0,0))
par(mfrow=c(3,3))
set.seed(2025)
PA_one_simulation = network_simulation("PA",TRUE)

PA_iterations = c(); PA_degrees = c()
for (simulation in 1:500)
{
  PA_simulation = network_simulation("PA",FALSE)
  PA_iterations = c(PA_iterations,length(PA_simulation$Iterations))
  PA_degrees = c(PA_degrees,PA_simulation$Degrees)
}

p1 = plot_evolution(PA_one_simulation$Iterations,PA_one_simulation$Real,
PA_one_simulation$Theory,"Evolution of the rumor")
p2 = plot_distribution(PA_iterations,c("Number of generations",
"Number of simulations","Number of generations for rumor transmission"))
p3 = plot_distribution(PA_degrees,c("Degrees","Number of vertices","Degree distribution"))
plot = arrangeGrob(p2,p3,ncol=2)
ggsave(filename="NW8.png",plot=p1,width=8,height=5,path="Figures")
ggsave(filename="NW9.png",plot=plot,width=12,height=6,path="Figures")
```

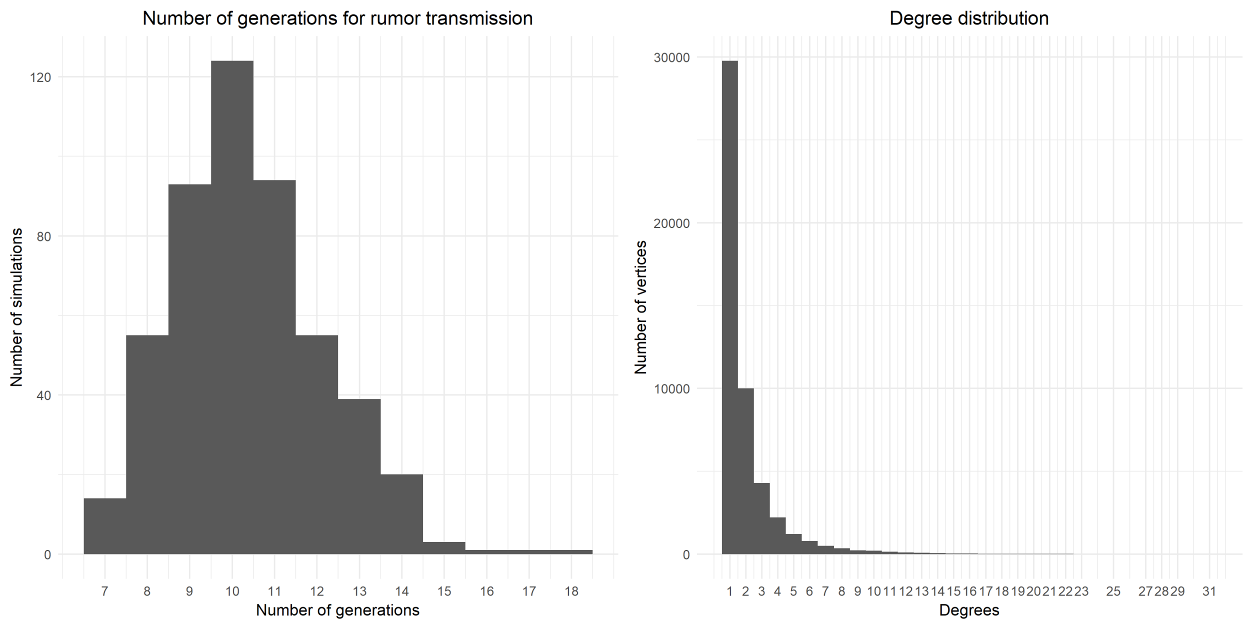
This figure shows the rumor propagation of a Preferential Attachment model simulation.



This figure shows the evolution in time for such rumor propagation.



This figure includes some metrics from 500 Preferential Attachment model simulations.



The transmission of the rumor is complete (all the individuals receive the rumor). The explanation to this is that all vertices are connected to the network so the rumor can reach them all. The transmission of the rumor is generally slow, with a mean of 10 generations, and high variability (the clustered structure, where groups of vertices are highly connected between them but isolated from others, makes the cluster in which the rumor starts highly decisive). The degree distribution resembles a power low distribution, where most vertices have low degrees and few vertices have high degrees. This is coherent with the process of building the edges.

It is quite obvious that the results we obtained depend a lot on the parameters we have chosen. We used 1 as the power of preferential attachment; increasing this metric would make the difference between popular and unpopular vertices even greater (very few individuals would gather most of the connections). This could turn into a very efficient propagation method (since everyone is connected to the hub, the rumor can rapidly get to it; since the hub is connected to everyone, the rumor can be widely spread).

6 Other excercises

These are the excercises I was not able to do.

6.1 Analyze an epidemic belonging to fiction movie

It actually looked like a fun exercise but I just had no time. However, I have one suggestion. If you restart this course at any point (you should, by the way), you could maybe update the list of movies. Here you have my recommendation: 'Dead Snow' or, as it has been translated to Spanish, 'Zombis Nazis'. I know what you might be thinking; 'Oh my god, what an awful combination!'. But trust me, it is absolutely brilliant; I couldn't laugh more. If you think this is too much, another option is 'Zombieland'. You have probably seen this film, it turns a zombie apocalypse into something rather funny, which I kind of like.

6.2 Study the synchronization properties of a biological system

We have already seen things alike in synthetic biology courses so I preferred to explore other topics.

6.3 Study the dynamics of a population of replicators in neutral networks

I was quite in my own world during this class (we had received some bad news regarding our internships and it was just not a good day). In light of that, I decided to put my efforts in other topics.

6.4 Study the dynamics of viral infection within an individual

I think I am running out of excuses... Time again? No really, I went as far as I could and I preferred to do less exercises and understand them well.