Introducción a los modelos compartimentales

In this notebook, you will practice implementing compartmental models in R. This notebook is divided into two parts. The first part focuses on implementing SIS and SIR models using methods for solving differential equations. The second part focuses on implementing an agent-based model to simulate SIS and SIR dynamics.

Installing and Loading Libraries If you do not have the following libraries installed, run the following code block to install them.

```
#install.packages(c("tidyverse","deSolve","ggplot2"))
```

Load the libraries we will be using.

```
library(tidyverse)
library(deSolve)
library(ggplot2)
```

Implementing SIS and SIR Models with Differential Equations Let's start with the SIS model. Just as a reminder, this model is described by the following differential equations:

$$\frac{dS}{dt} = -\frac{\beta SI}{N} + \mu I$$
$$\frac{dI}{dt} = \frac{\beta SI}{N} - \mu I$$

Where *beta* corresponds to the force of infection, *mu* corresponds to the infectious period, and N corresponds to the total population.

To solve these equations, we will use the **deSolve** library. Within this package, we will use the **ode** function. This function looks like the following:

```
ode(y, times, func, parms,
    method = c("lsoda", "lsode", "lsodes", "lsodar", "vode", "daspk",
    "euler", "rk4", "ode23", "ode45", "radau",
    "bdf", "bdf_d", "adams", "impAdams", "impAdams_d", "iteration"),
    ...)
```

In general, to define this function, we need to know the initial values of the state variables y, the time series (*times*) for which we want a solution, and the function (*func*) that we want to solve. To input this function, we need to write a program that represents the different elements of the SIS model. The implementation would be as follows:

```
sis.model<-function(t,x,parameters){
  #Extract the state variables
  S <- x[1]
  I <- x[2]

  #Extract the parameters
  beta <- parameters["beta"]
  mu <- parameters["mu"]
  N <- S + I
```

```
  #Now the code that models the equations

  dSdt <- -((beta*S*I)/N) + mu*I #Equation for susceptibles
  dIdt <- ((beta*S*I)/N) - mu*I #Equation for infected

  #Combine the results in a unitary vector

  dxdt <- c(dSdt,dIdt)

  #Return the results as a list
  list(dxdt)
}
```

With the function created, we can define the arrays that are going to store the initial conditions information

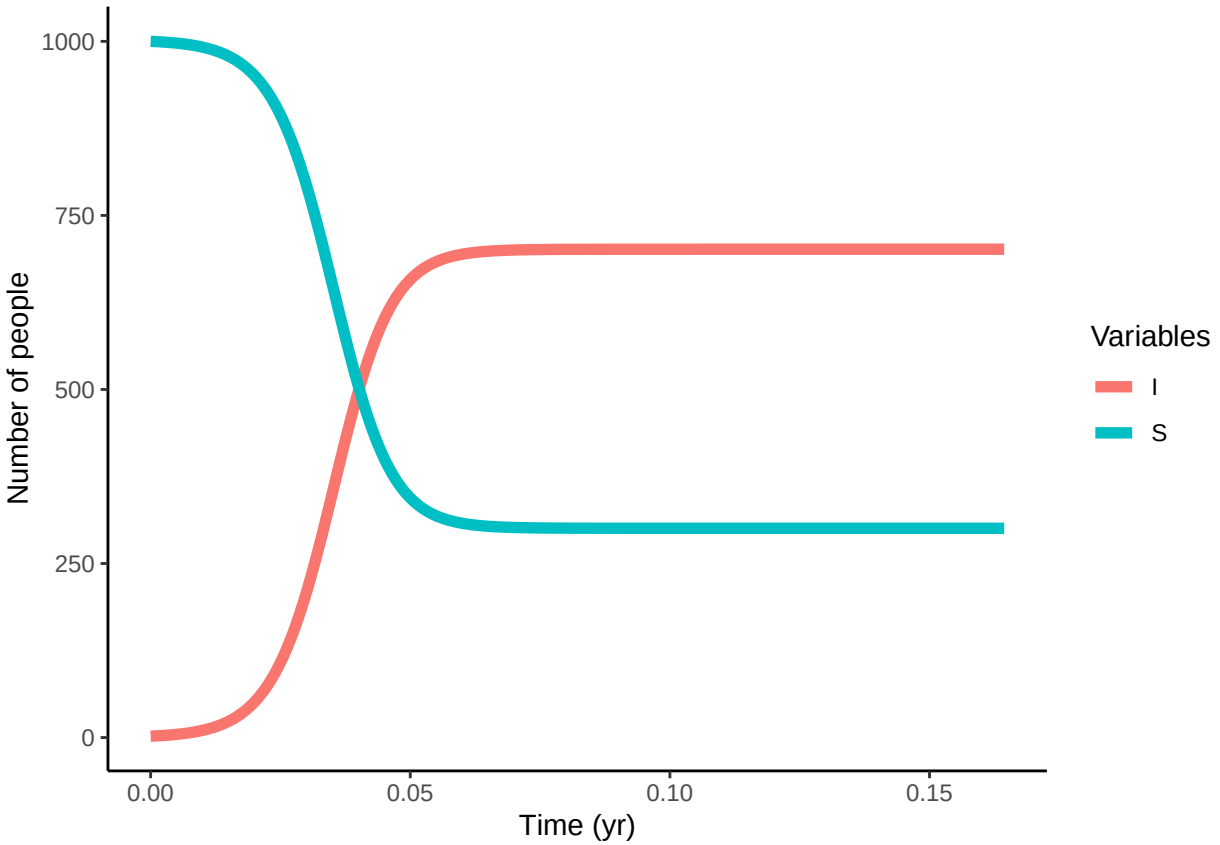
```
parameters <- c(beta = 250, mu = 75) #Epidemic parameters
times <- seq(from=0,to=60/365,by=1/365/4) #Timeline to evaluate
xstart <- c(S = 1000,I = 2) #Initial parameters for the populations
```

With the function and parameters defined, we will only use the function **ode** to solve the created function. We are going to store the information in a file called **output**

```
ode(
  func = sis.model,
  method = "euler",
  y = xstart,
  times = times,
  parms = parameters,
)%>%
  as.data.frame() -> output
```

In the file **output**, we can see three columns: the first one with time, the second one with the values of t for the susceptible population, and the third one with the values of t for the infected population. Now, let's plot the results of our curves:

```
p<- output %>%
  gather(variable,value,-time) %>%
  ggplot(aes(x=time,y=value,color=variable))+
  geom_line(linewidth=2)+
  theme_classic()+
  labs(x='Time (yr)',y='Number of people',color = "Variables")
p
```

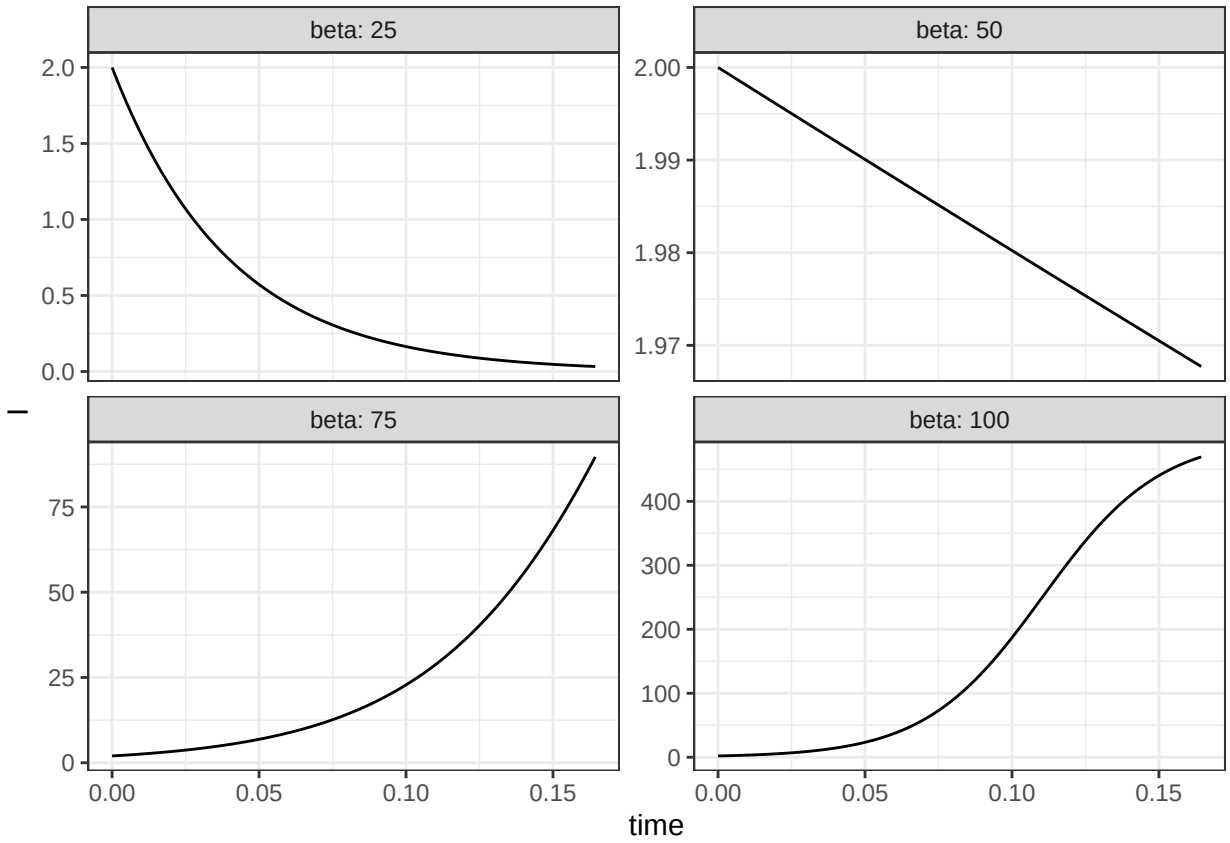


Let's see how we can graph different figures in a same grid using different values of *beta* and *mu*. We will start generating different values for *beta* and *mu*

```
values_beta <- c(25,50,75,100)
values_mu <- 50
```

Now let's combine these vectors along with the function **ode** to solve every combination of parameters:

```
expand.grid(beta=values_beta,mu=values_mu)%>%
  group_by(beta) %>%
  do(
    {
      ode(func=sis.model,y=xstart,times=times,
        parms=c(beta=.$beta,mu=.$mu)) %>%
        as.data.frame()
    }
  ) %>%
  ggplot(aes(x=time,y=I))+
  geom_line()+
  facet_wrap(~beta,scales='free_y',labeller=label_both)+
  theme_bw()
```

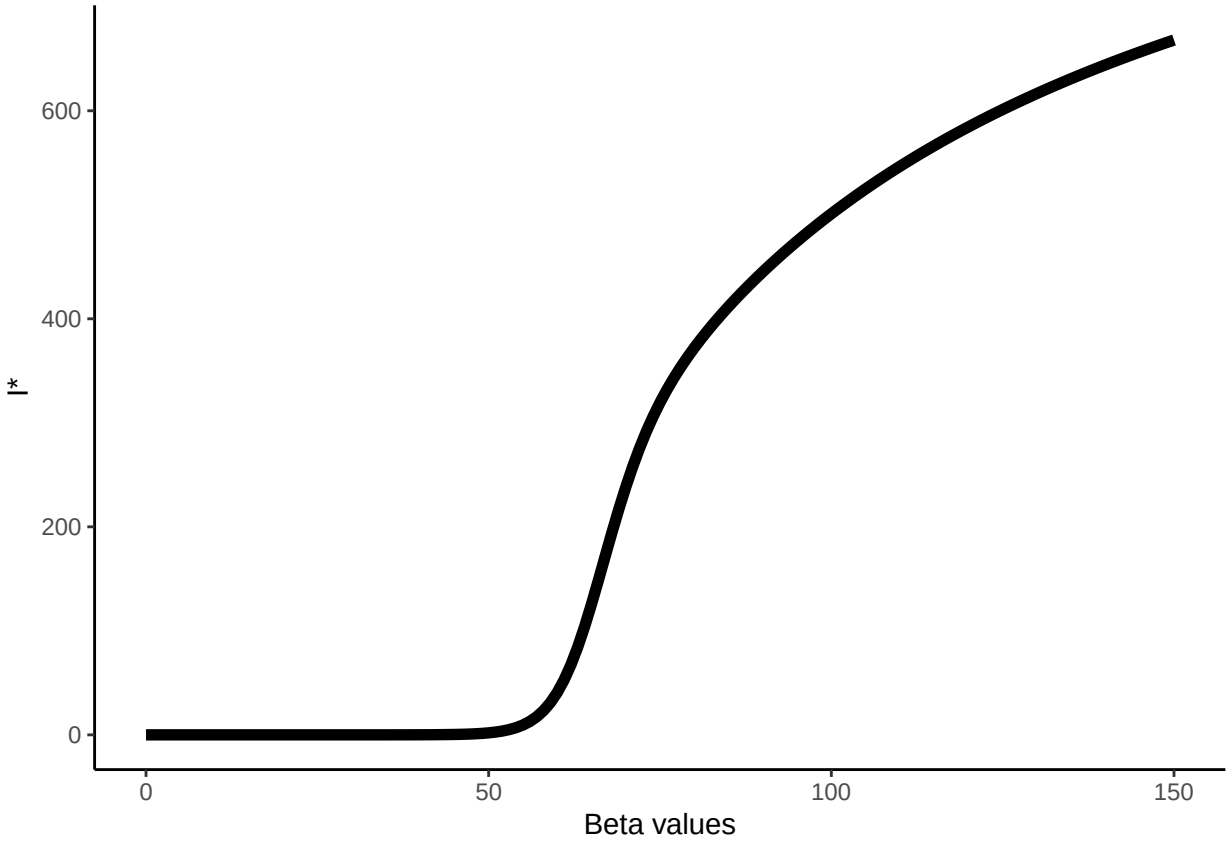


Anything under beta 50 decreases and anything above increases, this is our state where no changes are present.

In this graph, we can see how the curves of infected individuals behave as the parameters change. Now, let's plot how the number of infected individuals changes when the system stabilizes under different variations of the parameter mu.

```
betas <- seq(from=0,to=150,by =1)
mus <- rep(50,length(betas))
Istable <- rep(NA,length(betas))

for(i in 1:length(betas)){
  parameters <- c(beta = betas[i], mu = mus[i]) #Epidemic parameters
  times <- seq(from=0,to=120/365,by=1/365/4) #Timeline to evaluate
  xstart <- c(S = 1000,I = 2) #Initial parameters for the populations
  ode(
    func = sis.model,
    method = "euler",
    y = xstart,
    times = times,
    parms = parameters,
  )%>%
    as.data.frame() -> output
  Istable[i]<-output$I[nrow(output)]
}

output<-data.frame(betas,Istable)
```

```r
ggplot(data = output,aes(x=betas,y=Istable))+
  geom_line(linewidth=2)+
  theme_classic()+
  labs(x='Beta values',y='I*')
```



Now try to build on your own the function for a SIR model. Remember that the formulas for the SIR are the following:

$$\frac{dS}{dt} = -\frac{\beta SI}{N}$$
$$\frac{dI}{dt} = \frac{\beta SI}{N} - \mu I$$
$$\frac{dR}{dt} = \mu I$$

Use the following space in the code to build the function:

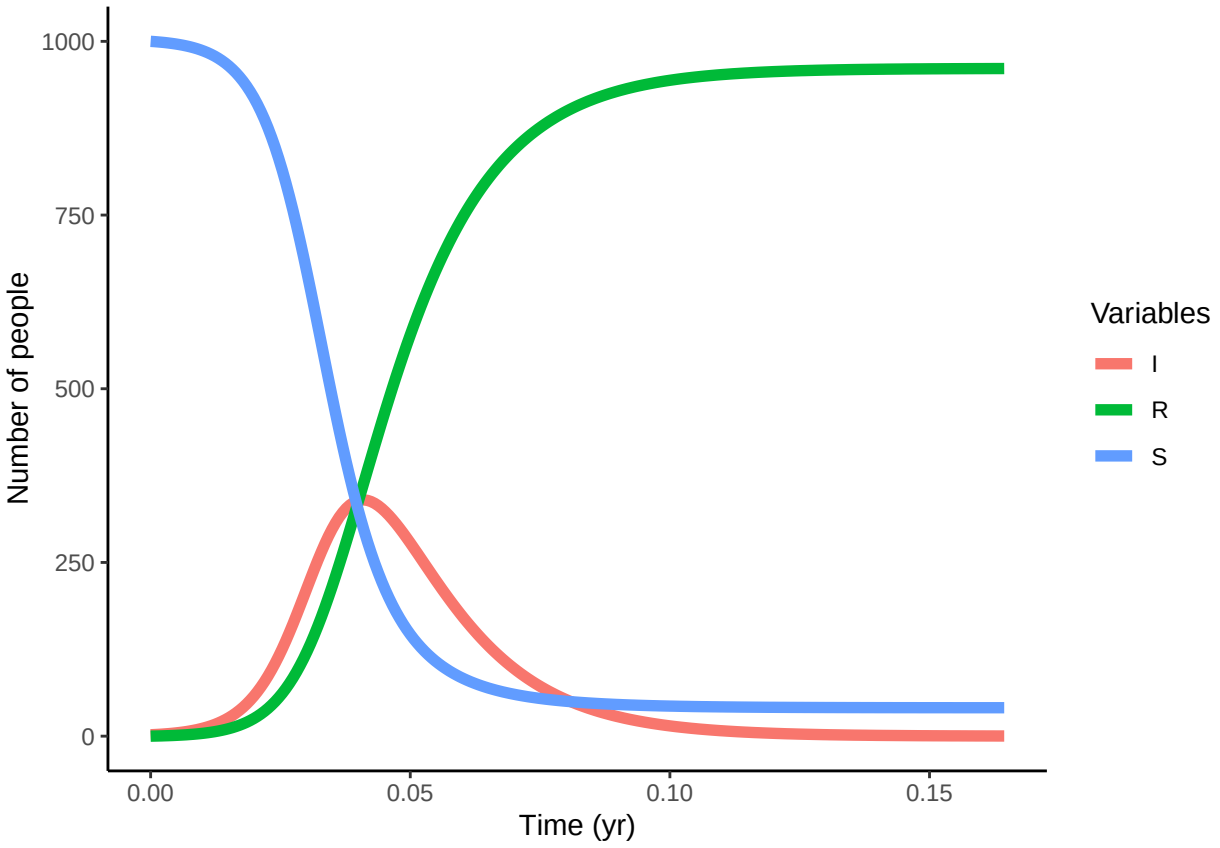
```r
sir.model<-function(t,x,parameters){
  #Extract all the state variables
  S <- x[1]
  I <- x[2]
  R <- x[3]

  #Extract the parameters
  beta <- parameters["beta"]
```

```
  mu <- parameters["mu"]
  N <- S + I + R

  #Now the code that models the equations
  dSdt <- -((beta*S*I)/N)  #Equation for susceptibles
  dIdt <- ((beta*S*I)/N) - mu*I #Equation for infected
  dRdt <- mu*I
  dxdt <- c(dSdt,dIdt,dRdt)

  #Return the results as a list
  list(dxdt)
}
```

When you have implemented the function, run the next block to see the results with some initial parameters:

```
parameters <- c(beta = 250, mu = 75) #Epidemic parameters
times <- seq(from=0,to=60/365,by=1/365/4) #Timeline to evaluate
xstart <- c(S = 1000,I = 2, R=0) #Initial parameters for the populations

ode(
  func = sir.model,
  y = xstart,
  times = times,
  parms = parameters,
)%>%
  as.data.frame() -> output2

output2 %>%
  gather(variable,value,-time) %>%
  ggplot(aes(x=time,y=value,color=variable))+
  geom_line(size=2)+
  theme_classic()+
  labs(x='Time (yr)',y='Number of people',color = "Variables")
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```




```
rm(output2,output,parameters,values_beta,values_mu,xstart,
   sis.model,sir.model)
```

Using agent models to simulate SIS and SIR dynamics The first aspect to consider in agent-based modeling is that we have individuals, initially homogeneous (in subsequent lectures we will include more heterogeneity). To model our agents in a simpler way, we will separate the processes of infection and recovery. In this model, we will assume that each agent has two attributes: a unique identifier and a state (0 for susceptible and 1 for infected). At each time step t, infected agents can have *k_contacts* with susceptible agents, and in that contact, they can transmit the infection with a probability *lambda*.

For the function that will allow us to model the infection, we will use as parameters the database that stores the state of the agents (*df*), the number of contacts that each infected agent will have (*contacts*), and the infection probability (*lambda*).

Data frame: id of the individual, the state (0,1) -> not infected, infected. There's a random chance of getting infected and the same possibility for every individual to infect others and themselves but depends on the changing states.

```
infect <- function(df, contacts, lambda){
  n_infected <- nrow(df[df$state==1,]) #Obtain the number of infected
  #Obtain the random sample of agents to get infected
  sample_to_infect <- df[df$id%in%sample(df$id[df$state==0],
                              min(n_infected*contacts,nrow(df[df$state==0,])),replace = T),]
  random <- runif(nrow(sample_to_infect),0,1) #Generate as many random numbers as infected with probabi
  chance <- as.integer(random<=lambda) #it is evaluated if the random number is lower than lambda if it
  df$state[df$id%in%sample_to_infect$id]<-df$state[df$id%in%sample_to_infect$id]+chance#change the stat
```

```
  return(df)
}
```

Recovery occurs after infection and only happens to currently infected agents. Each infected agent at each time step t has a probability mu of recovering and becoming susceptible again. For this function, we will input the database that stores the state of the agents (df) and the recovery probability (mu) as parameters.

```
recover <- function(df,ids_infected,mu){
  random <- runif(length(ids_infected),0,1) #Generate as many random numbers as infected with the proba
  chance <- as.integer(random<=mu) #It is evaluated if the random number is lower than mu for recover,
  df$state[df$state==1 & df$id%in%ids_infected]<-df$state[df$state==1 & df$id%in%ids_infected]-chance #
  return(df)
}
```

Now we are going to create a function to recreate each moment t in time.

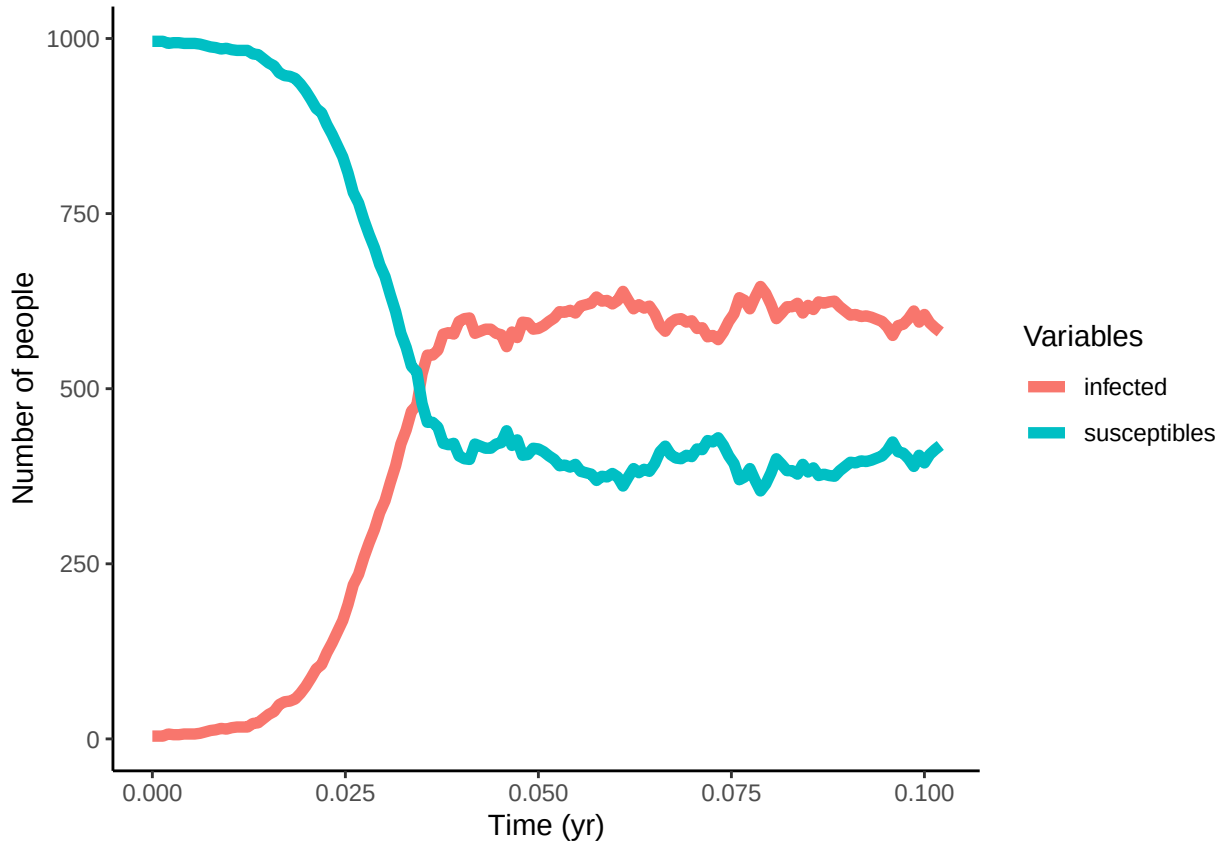
```
step<-function(df,lambda,mu,contacts){
  ids_infected<-df$id[df$state==1]
  df<-infect(df,contacts,lambda)
  df<-recover(df,ids_infected,mu)
  return(df)
}
```

Finally, we create the routine to simulate each day. Initially, we will run only one instance of the model. However, the idea is to be able to run multiple instances of the same model to obtain a confidence interval for the curves.

```
N <- 1000 #Initial agent population
T <- 150 #Intervals of time to simulate
n_infected_initial <- 4 #Number of initial infected agents
df <- data.frame(id = 1:N,state = rep(0,N))#We create the dataframe with the initialization of the iden

#Assign the initial infected

df$state[df$id%in%sample(df$id,n_infected_initial)]<-1

#Define the model parameters
lambda <- 0.2
mu <- 0.08
contacts <- 1

#Create the lists to store the information of susceptibles and infected

infected<-rep(NA,T)
susceptibles<-rep(NA,T)

for(i in 1:T){
  #Store the number of susceptible and infected in the lists
  infected[i] <- nrow(df[df$state==1,])
  susceptibles[i] <- nrow(df[df$state==0,])
  df<-step(df,lambda,mu,contacts) #Generate a step in the model
}
```

Graph our curves:

```
output <-data.frame(time = times[1:length(susceptibles)],susceptibles,infected)

q<-output %>%
  gather(variable,value,-time) %>%
  ggplot(aes(x=time,y=value,color=variable))+
  geom_line(size=2)+
  theme_classic()+
  labs(x='Time (yr)',y='Number of people',color = "Variables")
q
```



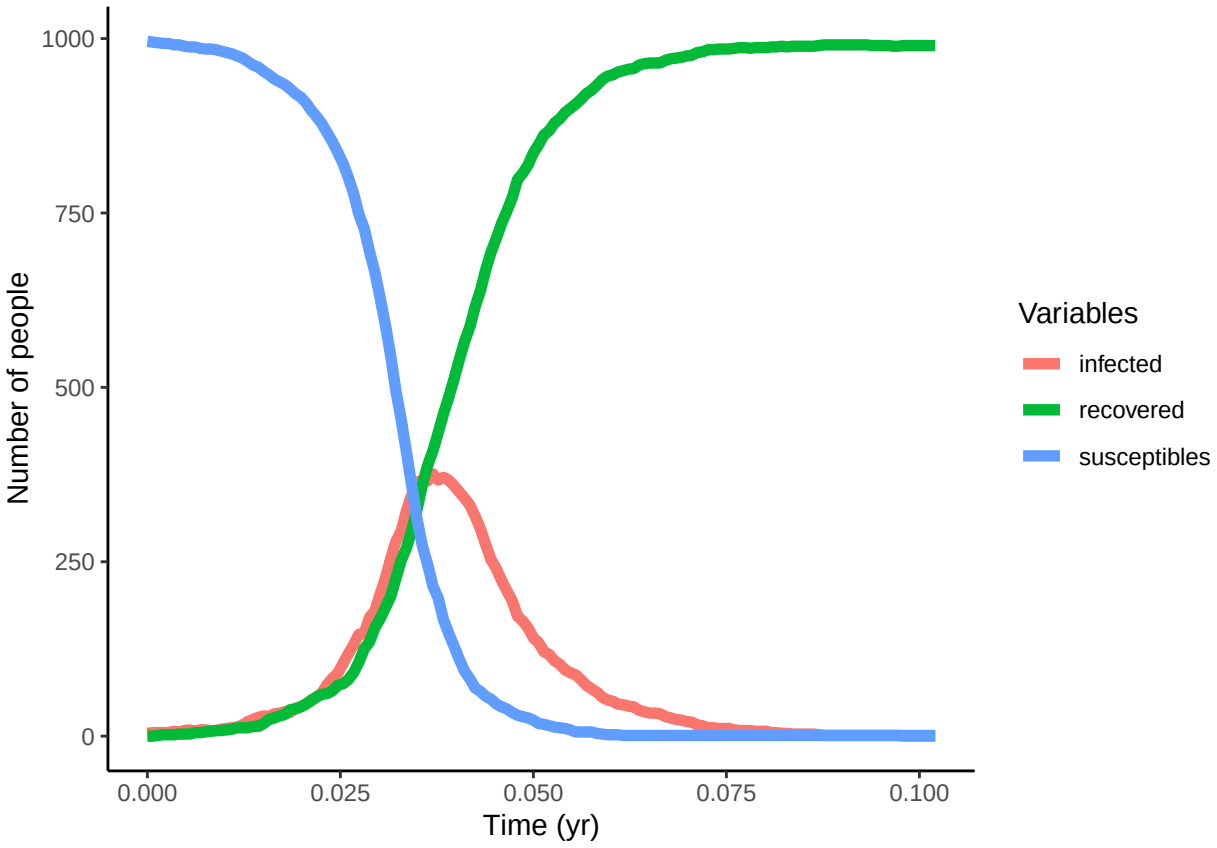
Now try to replicate on your own the SIR model with the agent model.

```
infect <- function(df, contacts, lambda){
  n_infected <- nrow(df[df$state==1,]) #Obtain the number of infected
  #Obtain the random sample of agents to get infected
  sample_to_infect <- df[df$id%in%sample(df$id[df$state==0],
                                         min(n_infected*contacts,nrow(df[df$state==0,])),replace = T),]
  random <- runif(nrow(sample_to_infect),0,1) #Generate as many random numbers as infected with probabi
  chance <- as.integer(random<=lambda) #it is evaluated if the random number is lower than lambda if it
  df$state[df$id%in%sample_to_infect$id]<-df$state[df$id%in%sample_to_infect$id]+chance#change the stat
  return(df)
}
```

```r
recover <- function(df,ids_infected,mu){
  random <- runif(length(ids_infected),0,1) #Generate as many random numbers as infected with the proba
  chance <- as.integer(random<=mu) #It is evaluated if the random number is lower than mu for recover,
  df$state[df$state==1 & df$id%in%ids_infected]<-df$state[df$state==1 & df$id%in%ids_infected]+chance #
  return(df)
}
```

```r
N <- 1000 #Initial agent population
T <- 150 #Intervals of time to simulate
n_infected_initial <- 4 #Number of initial infected agents
df <- data.frame(id = 1:N,state = rep(0,N))#We create the dataframe with the initialization of the iden

#Assign the initial infected

df$state[df$id%in%sample(df$id,n_infected_initial)]<-1

#Define the model parameters
lambda <- 0.2
mu <- 0.08
contacts <- 1

#Create the lists to store the information of susceptibles and infected

infected<-rep(NA,T)
susceptibles<-rep(NA,T)
recovered<-rep(NA,T)

for(i in 1:T){
  #Store the number of susceptible and infected in the lists
  infected[i] <- nrow(df[df$state==1,])
  susceptibles[i] <- nrow(df[df$state==0,])
  recovered[i] <- nrow(df[df$state==2,])
  df<-step(df,lambda,mu,contacts) #Generate a step in the model
}
```

```r
output <-data.frame(time = times[1:length(susceptibles)],susceptibles,infected,recovered)

q<-output %>%
  gather(variable,value,-time) %>%
  ggplot(aes(x=time,y=value,color=variable))+
  geom_line(size=2)+
  theme_classic()+
  labs(x='Time (yr)',y='Number of people',color = "Variables")
q
```



Instructions for the final work For the final work, you must include in an RMarkdown script both types of simulations for the SIR model. For each simulation, you should include the following elements:

- The script with the computational code that allows simulating the SIR model.
- A plot that shows the results of the SIR simulation.
- Some paragraphs including conclusions from the exercise and the main differences found compared to the SIS model in terms of modeling.

Review of random networks and Markov models

In this notebook, you will be able to review the generation of different random networks and practice implementing Markov models that allow you to simulate infections that work with the adjacency matrices we generate

Installing and loading libraries If you do not have the following libraries installed, run the following code block to install them.

```
#install.packages(c("deSolve","sna","igraph","network","ggnet2", "intergraph", "GGally"))
```

Load the libraries that we are going to use

```r
library(tidyverse)
library(igraph)
library(sna)
library(network)
library(ggplot2)
library(GGally)
```

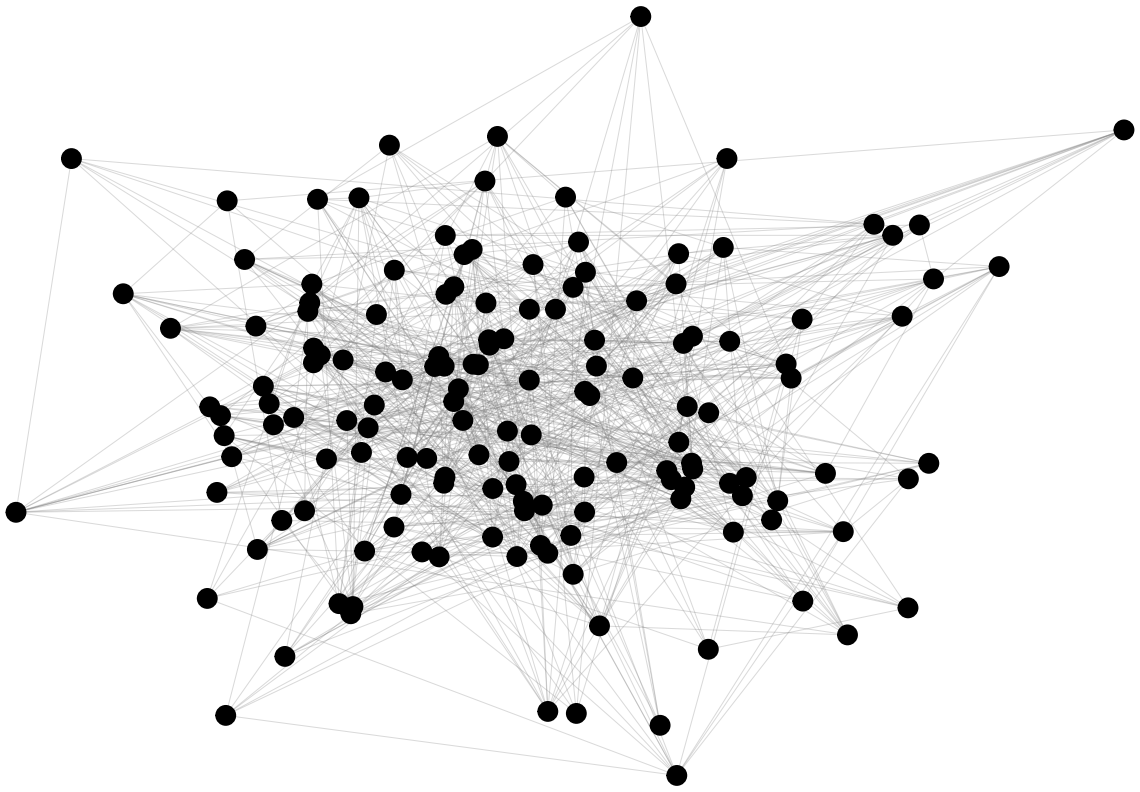
Generating random networks Let's start with the most common networks, starting with the Erdos-Renyi network. For this, we will use the igraph library with its erdos.renyi.game function. This function is as follows:

```r
erdos.renyi.game(
  n,
  p.or.m,
  type = c("gnp", "gnm"),
  directed = FALSE,
  loops = FALSE
)
```

It receives different parameters, but for this exercise we will only be interested in n, which corresponds to the number of nodes in the network, p, which is the probability of creating an edge between two nodes, and directed to configure whether the network is directed or undirected

Let's see how to create a network and graph it:

```r
g1 <- erdos.renyi.game(n = 150, p = 10 / 150)
ggnet2(g1,mode = "spring", vjust = -1, size = 3, color = 1,edge.alpha = 0.3,edge.size = 0.1)
```



Now we are going to extract the adjacency matrix from the network we just created. For that, we are going to use the as_adjacency_matrix function, which receives as a parameter the network from which the matrix will be extracted

```
A1 <- as.matrix(as_adjacency_matrix(g1))
```

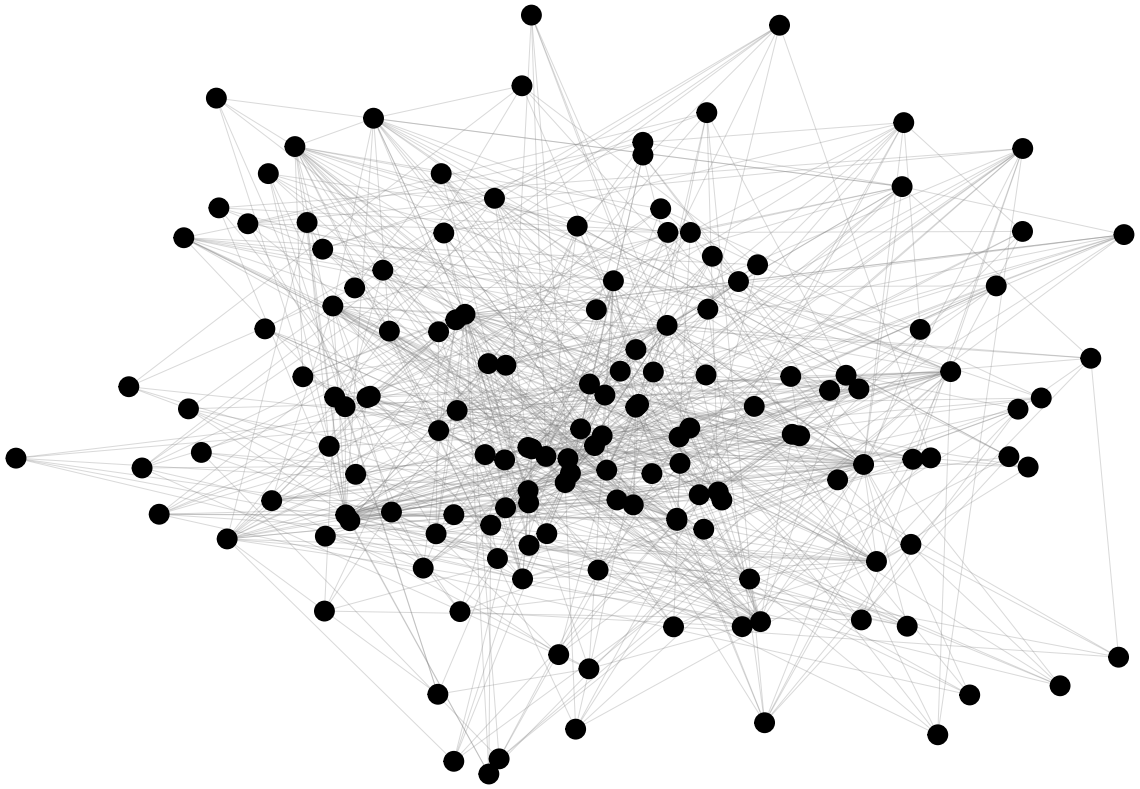
Now let's create a scale free network using the Barabasi-Albert model. The function to use will be **sample_pa**, which has the following form:

```
sample_pa(
  n,
  power = 1,
  m = NULL,
  out.dist = NULL,
  out.seq = NULL,
  out.pref = FALSE,
  zero.appeal = 1,
  directed = TRUE,
  algorithm = c("psumtree", "psumtree-multiple", "bag"),
  start.graph = NULL
)
```

This function receives different parameters, but for this tutorial we will only be interested in n, which corresponds to the number of nodes in the network, *power*, which is the power of the *preferential attachment* and by default is set to 1 (*linear preferential attachment*), and *directed* to configure whether the network is directed or undirected.

Let's see how to create a network and graph it:

```
g2 <- sample_pa(n = 150, m = 5, directed = FALSE)
A2 <- as.matrix(as_adjacency_matrix(g2))
ggnet2(g2,mode = "spring", vjust = -1, size = 3, color = 1,edge.alpha = 0.3,edge.size = 0.1)
```



Cascade-type contagion

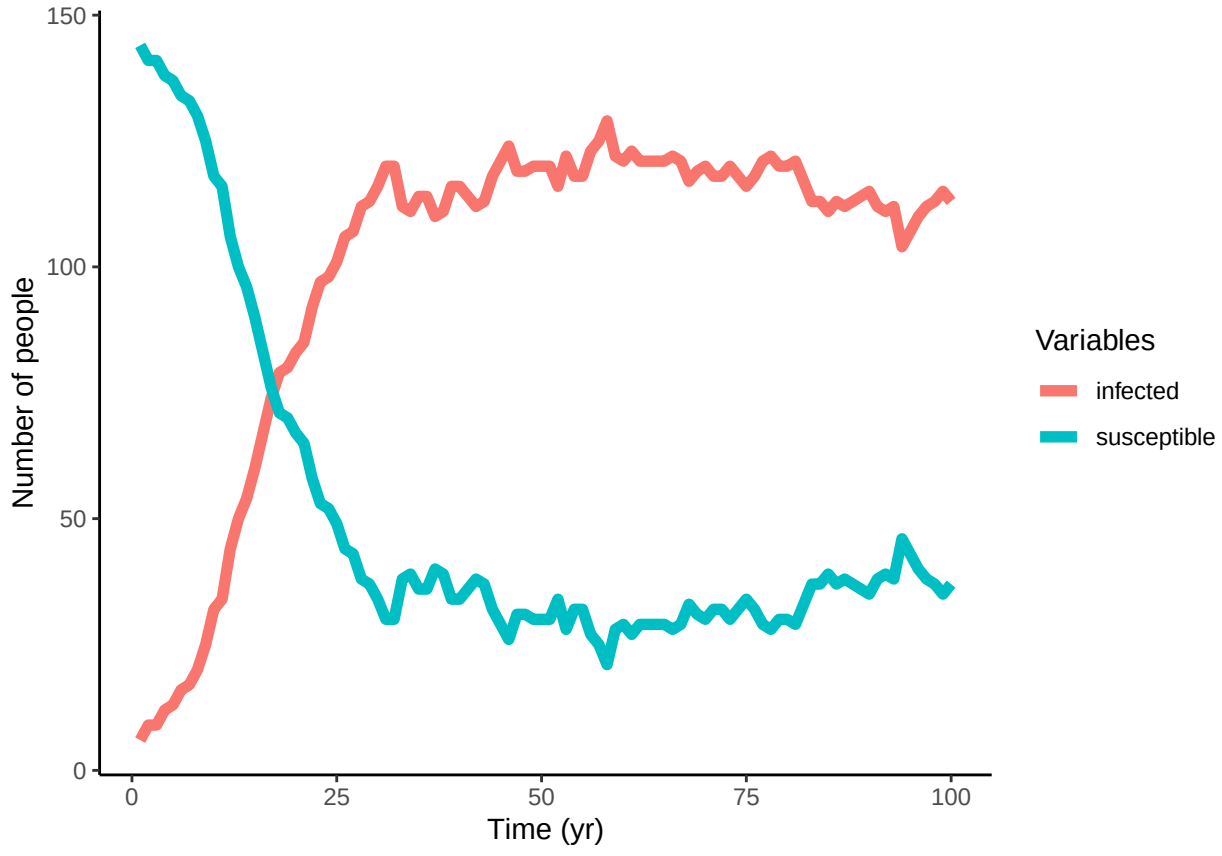
For the contagion, we will roll the dice for infection with all the neighbors of each node, using the adjacency matrix as a reference. Let's start by implementing the contagion for an SIS model

```
Tsim <- 100
lambda <- 0.02 #Contagion rate
gamma <- 0.035 #Recovery rate

#Lists to store the values for each t from infected and susceptible
susceptible <- rep(NA,Tsim)
infected <- rep(NA,Tsim)
state <- rbinom(150,1,5/150)
ids_nodes<-1:150

for(i in 1:Tsim){
  infected[i]<-sum(state==1) #Register the infected
  susceptible[i]<-sum(state==0) #Register the susceptible
```

```
  #Infection dynamics
  infected_nodes<-ids_nodes[state==1] #Extract the infected nodes
  for(j in 1:length(infected_nodes)){
    neighbours <- ids_nodes[A1[infected_nodes[j],]==1] #Extract the neighbours of each infected node
    if(length(neighbours)>0){
      random <- runif(length(neighbours),0,1)
      chance <- as.integer(random<=lambda)
      state[neighbours]<-ifelse(state[neighbours]==0,state[neighbours]+chance,state[neighbours])
    }
  }
  #Recovery dynamics
  random <- runif(length(infected_nodes),0,1)
  chance <- as.integer(random<=gamma)
  state[infected_nodes]<-state[infected_nodes]-chance
}
```

Let's graph the behaviour of the curve:

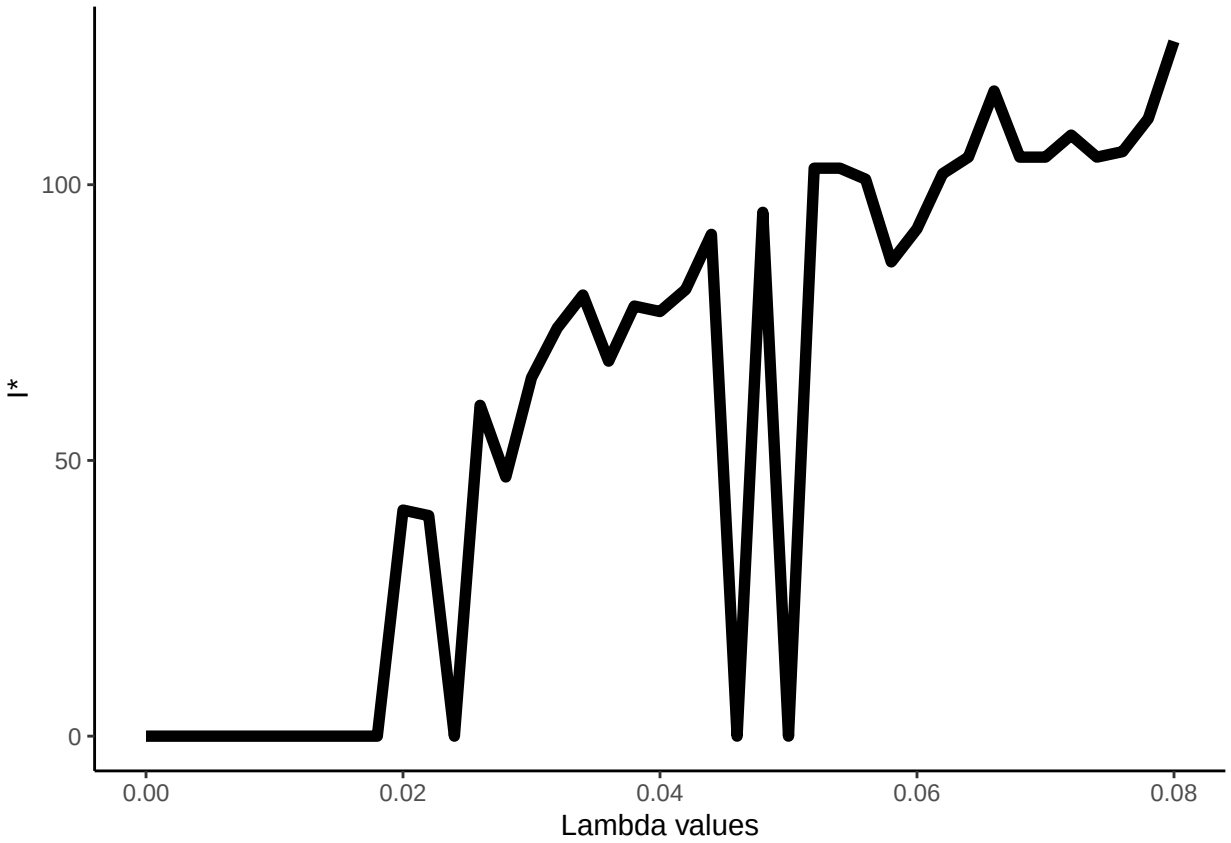
```
output <-data.frame(time = 1:length(susceptible),susceptible,infected)
q<-output %>%
gather(variable,value,-time) %>%
ggplot(aes(x=time,y=value,color=variable))+
geom_line(linewidth=2)+
theme_classic()+
labs(x='Time (yr)',y='Number of people',color = "Variables")
q
```



Just as we did in the first notebook, we are going to plot how the number of infected individuals changes when the system stabilizes under different variations of lambda.

```
lambdas<-seq(from=0,to=0.08,by =0.002)
gammas<-rep(0.15,length(lambdas))
Istable <- rep(NA,length(lambdas))
```

```
for(t in 1:length(lambdas)){

  Tsim <- 250
  lambda <- lambdas[t] #Contagion rate
  gamma <- gammas[t] #Recovery rate

  #Lists to store the values for each t from the infected and susceptible
  susceptible <- rep(NA,Tsim)
  infected <- rep(NA,Tsim)
  state <- rbinom(150,1,5/150)
  ids_nodes<-1:150

  for(i in 1:Tsim){

    infected[i]<-sum(state==1) #Register the infected
    susceptible[i]<-sum(state==0) #Register the susceptible

    #Infection dynamics
    infected_nodes<-ids_nodes[state==1] #Extract the infected nodes
    if(length(infected_nodes)>0){
      for(j in 1:length(infected_nodes)){

        neighbours <- ids_nodes[A1[infected_nodes[j],]==1] #Extract the neighbours of each infected nod
        if(length(neighbours)>0){

          random <- runif(length(neighbours),0,1)
          chance <- as.integer(random<=lambda)
          state[neighbours]<-ifelse(state[neighbours]==0,state[neighbours]+chance,state[neighbours])
        }
      }
    }

    #Recovery dynamics
    random <- runif(length(infected_nodes),0,1)
    chance <- as.integer(random<=gamma)
    state[infected_nodes]<-state[infected_nodes]-chance
  }
  Istable[t]<-infected[length(infected)]
}

I_graph<-Istable
output<-data.frame(lambdas,Istable)

ggplot(data = output,aes(x=lambdas,y=Istable))+
  geom_line(linewidth=2)+
  theme_classic()+
  labs(x='Lambda values',y='I*')
```



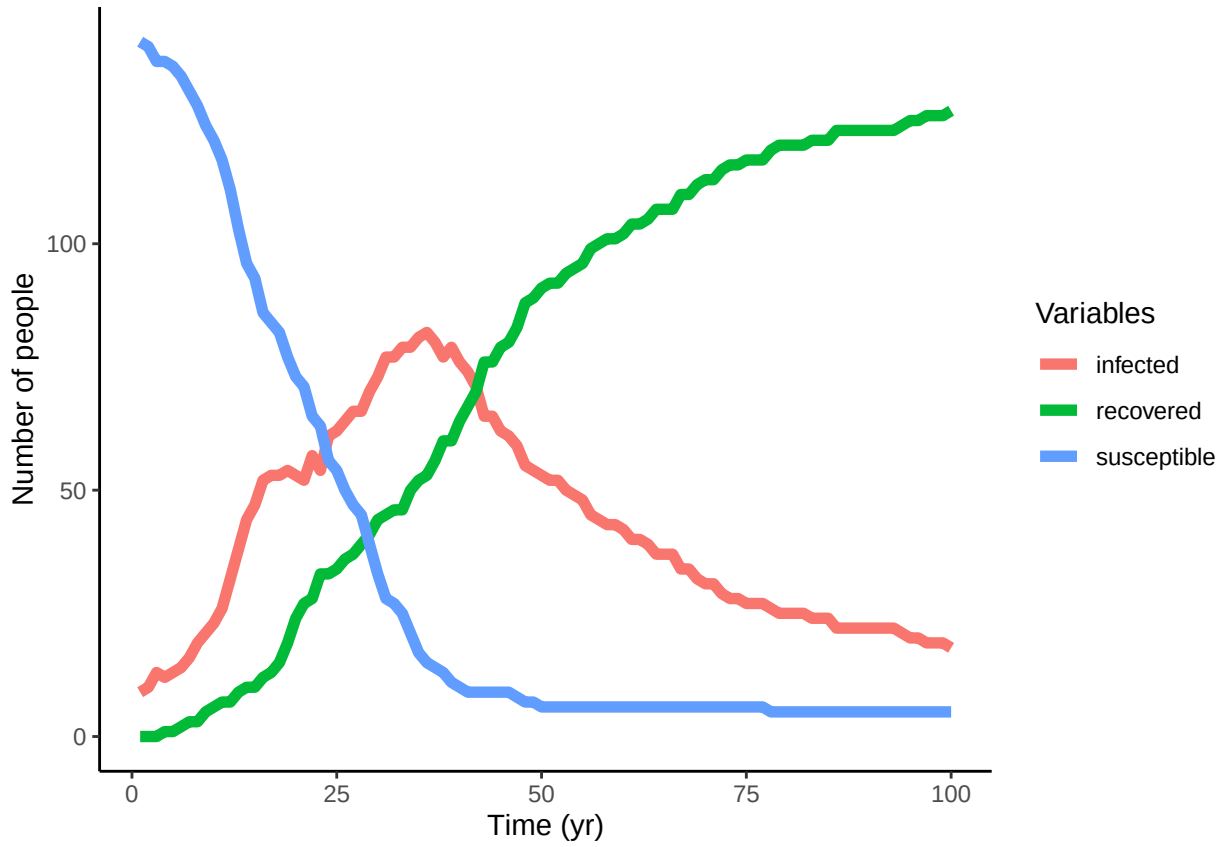
Exercise: now try to modify the two previous algorithms to obtain the plots for an SIR model.

```
Tsim <- 100
lambda <- 0.02 #Contagion rate
gamma <- 0.035 #Recovery rate

#Lists to store the values for each t from infected and susceptible
susceptible <- rep(NA,Tsim)
infected <- rep(NA,Tsim)
recovered <- rep(NA,Tsim)
state <- rbinom(150,1,5/150)
ids_nodes<-1:150

for(i in 1:Tsim){
  infected[i]<-sum(state==1) #Register the infected
  susceptible[i]<-sum(state==0) #Register the susceptible
  recovered[i]<-sum(state==2)

  #Infection dynamics
  infected_nodes<-ids_nodes[state==1] #Extract the infected nodes
  for(j in 1:length(infected_nodes)){
    neighbours <- ids_nodes[A1[infected_nodes[j],]==1] #Extract the neighbours of each infected node
    if(length(neighbours)>0){
      random <- runif(length(neighbours),0,1)
      chance <- as.integer(random<=lambda)
      state[neighbours]<-ifelse(state[neighbours]==0,state[neighbours]+chance,state[neighbours])
```

```
    }
  }
  #Recovery dynamics
  random <- runif(length(infected_nodes),0,1)
  chance <- as.integer(random<=gamma)
  state[infected_nodes]<-state[infected_nodes]+chance
}
```

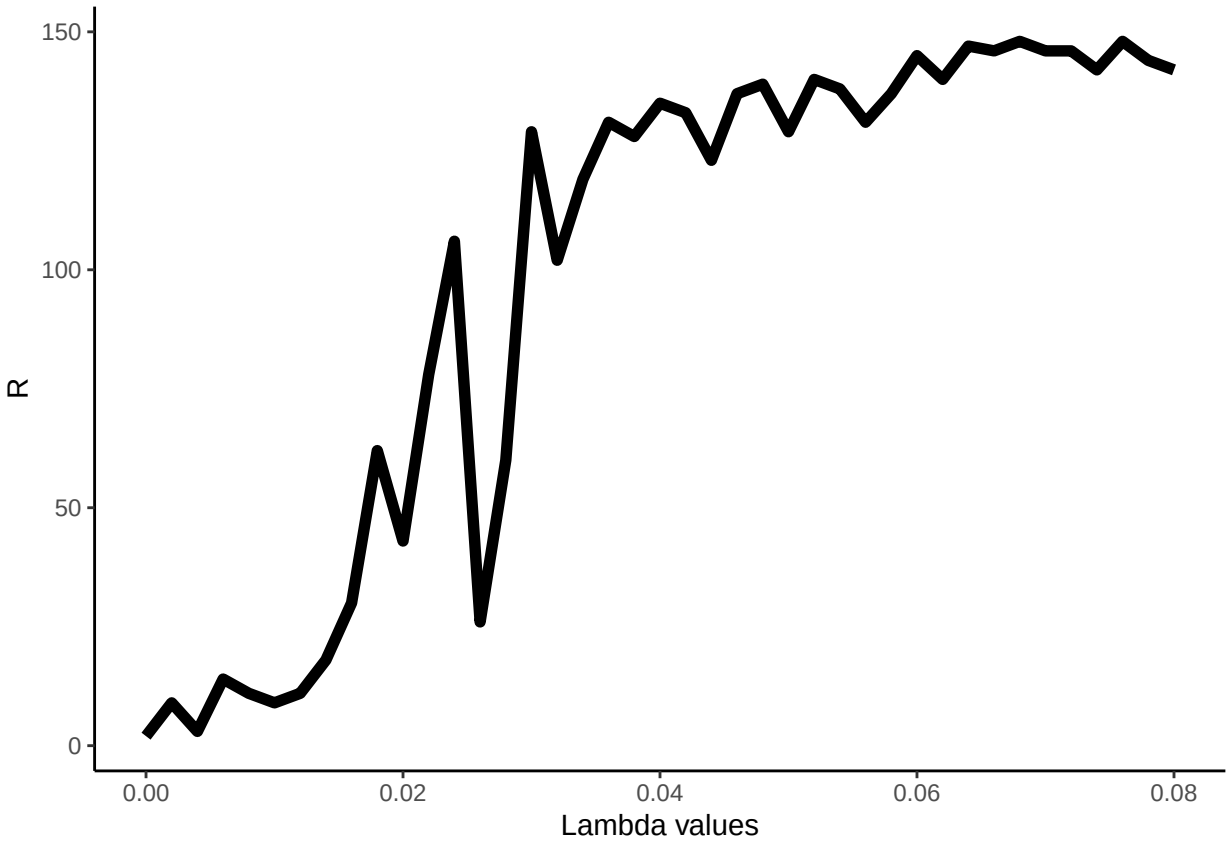
Let's graph the behaviour of the curve:

```
output <-data.frame(time = 1:length(susceptible),susceptible,infected,recovered)
q<-output %>%
gather(variable,value,-time) %>%
ggplot(aes(x=time,y=value,color=variable))+
geom_line(linewidth=2)+
theme_classic()+
labs(x='Time (yr)',y='Number of people',color = "Variables")
q
```




```
for(t in 1:length(lambdas)){

  Tsim <- 250
  lambda <- lambdas[t] #Contagion rate
  gamma <- gammas[t] #Recovery rate
```

```
  #Lists to store the values for each t from the infected and susceptible
  susceptible <- rep(NA,Tsim)
  infected <- rep(NA,Tsim)
  recovered <- rep(NA,Tsim)
  state <- rbinom(150,1,5/150)
  ids_nodes<-1:150

  for(i in 1:Tsim){

    infected[i]<-sum(state==1) #Register the infected
    susceptible[i]<-sum(state==0)
    recovered[i]<-sum(state==2)#Register the susceptible

    #Infection dynamics
    infected_nodes<-ids_nodes[state==1] #Extract the infected nodes
    if(length(infected_nodes)>0){
      for(j in 1:length(infected_nodes)){

        neighbours <- ids_nodes[A1[infected_nodes[j],]==1] #Extract the neighbours of each infected nod
        if(length(neighbours)>0){

          random <- runif(length(neighbours),0,1)
          chance <- as.integer(random<=lambda)
          state[neighbours]<-ifelse(state[neighbours]==0,state[neighbours]+chance,state[neighbours])
        }
      }
    }

    #Recovery dynamics
    random <- runif(length(infected_nodes),0,1)
    chance <- as.integer(random<=gamma)
    state[infected_nodes]<-state[infected_nodes]+chance
  }
  Istable[t]<-recovered[length(recovered)]
}

I_graph<-Istable
output<-data.frame(lambdas,Istable)

ggplot(data = output,aes(x=lambdas,y=Istable))+
  geom_line(linewidth=2)+
  theme_classic()+
  labs(x='Lambda values',y='R')
```



Markovian approach

Now let's try to model the same SIS model using the theoretical equations and compare them. For this, we will define two functions: $P_i(t)$ and $q_i(t)$. We have that P_i is defined as:

$$P_i(t+1) = P_i(t) * (1-\mu) + (1 - P_i(t)) * q_i(t)$$

defined as the probability of infection of node i for each time step t. On the other hand, the function q_i(t) is defined as:

$$q_i(t) = 1 - \prod_{j=1}^{n} [1 - \lambda A_{ij} P_j(t)]$$

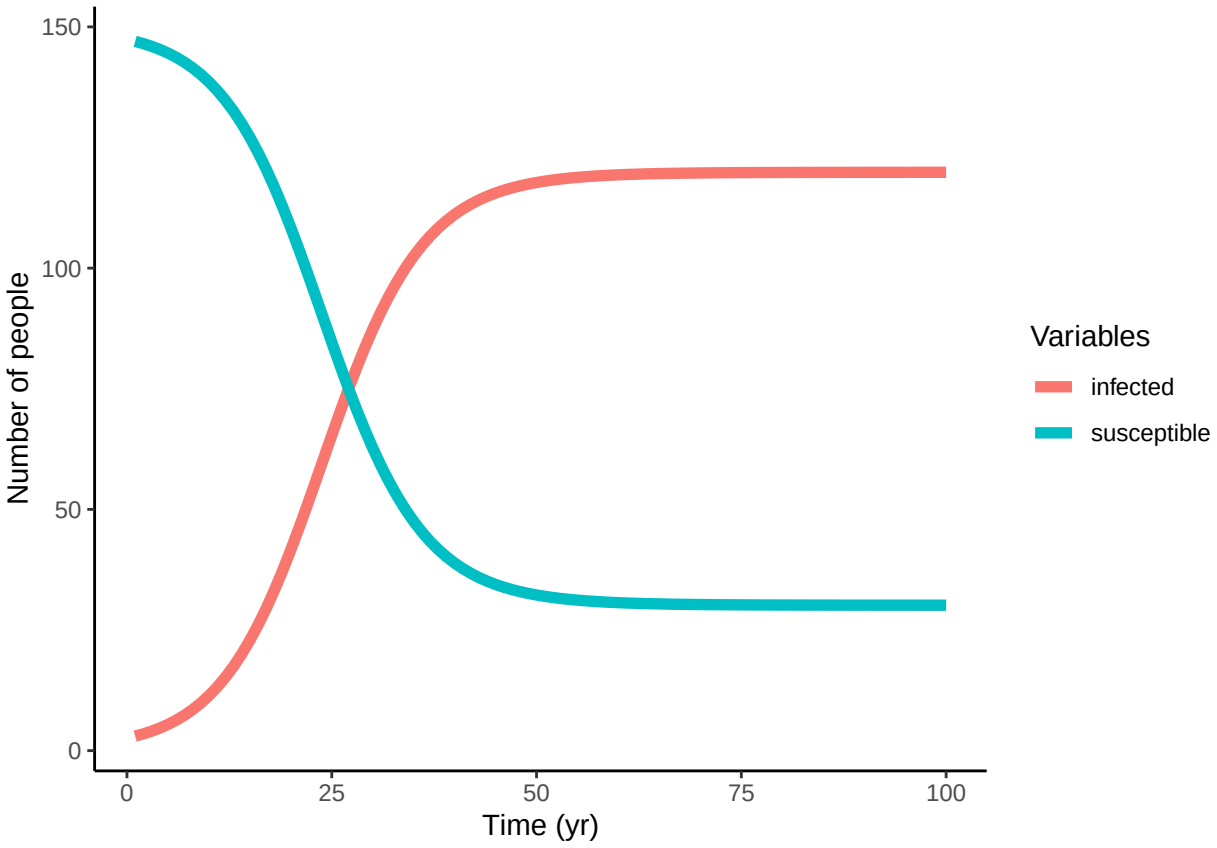
For the initial conditions of P_i(t), we equate all the values of the vector to the number of initial infected nodes divided by the total number of nodes. Let's start implementing these formulas for the SIS model

```
Tsim <- 100
lambda <- 0.02 #Contagion rate
mu <- 0.035 #Recovery rate

#Lists to store the values for each t from the infected and susceptible
```

```r
susceptible <- rep(NA,Tsim)
infected <- rep(NA,Tsim)
ids_nodes <-1:150

#P_i(t) and q_i(t)
q_i <- rep(0,150)

#Initial conditions for P_i(0)
P_i <- rep(3/150,150)

for(t in 1:Tsim){
  #Storing the infection percentages
  infected[t] <- sum(P_i)/150
  susceptible[t] <- 1-infected[t]
  for(i in 1:150){
    #Updating q_i(t)
    neighbours <- ids_nodes[A1[i,]==1]
    if(length(neighbours)>0){
      q_i[i]<-1
      for(j in 1:length(neighbours)){
        q_i[i]<-q_i[i]*(1-lambda*P_i[j])
      }
      q_i[i]<-1-q_i[i]
    }
  }
  #Updating P_i(t)
  P_i <- P_i*(1-mu)+(1-P_i)*q_i
}
rm(P_i,q_i,mu,lambda,i,ids_nodes,t,Tsim,neighbours,j)
```

Let's see the behaviour of the curves:

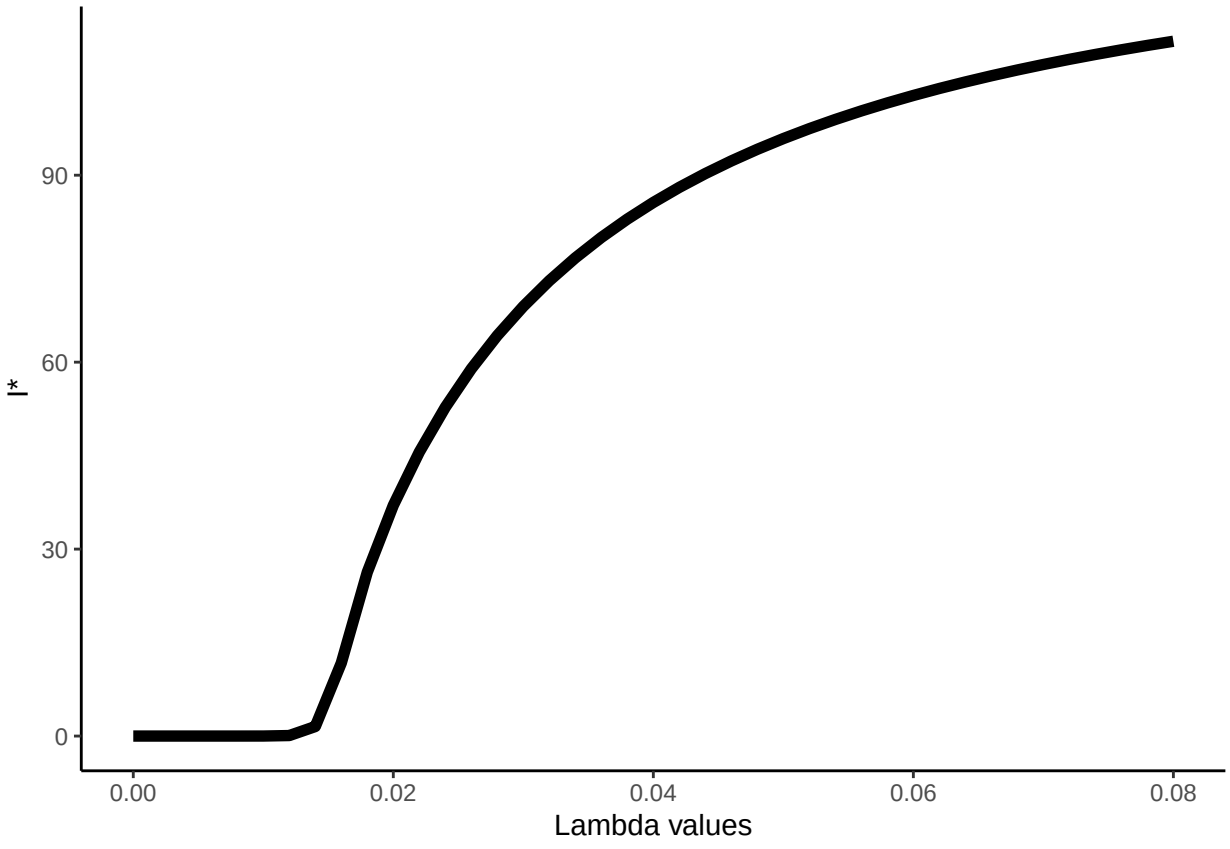
```r
output <-data.frame(time = 1:length(susceptible),susceptible = susceptible*150,infected = infected*150)
q<-output %>%
  gather(variable,value,-time) %>%
  ggplot(aes(x=time,y=value,color=variable))+
  geom_line(linewidth=2)+
  theme_classic()+
  labs(x='Time (yr)',y='Number of people',color = "Variables")
q
```



Just as we did above, we are going to plot how the number of infected individuals changes when the system stabilizes under different variations of lambda.

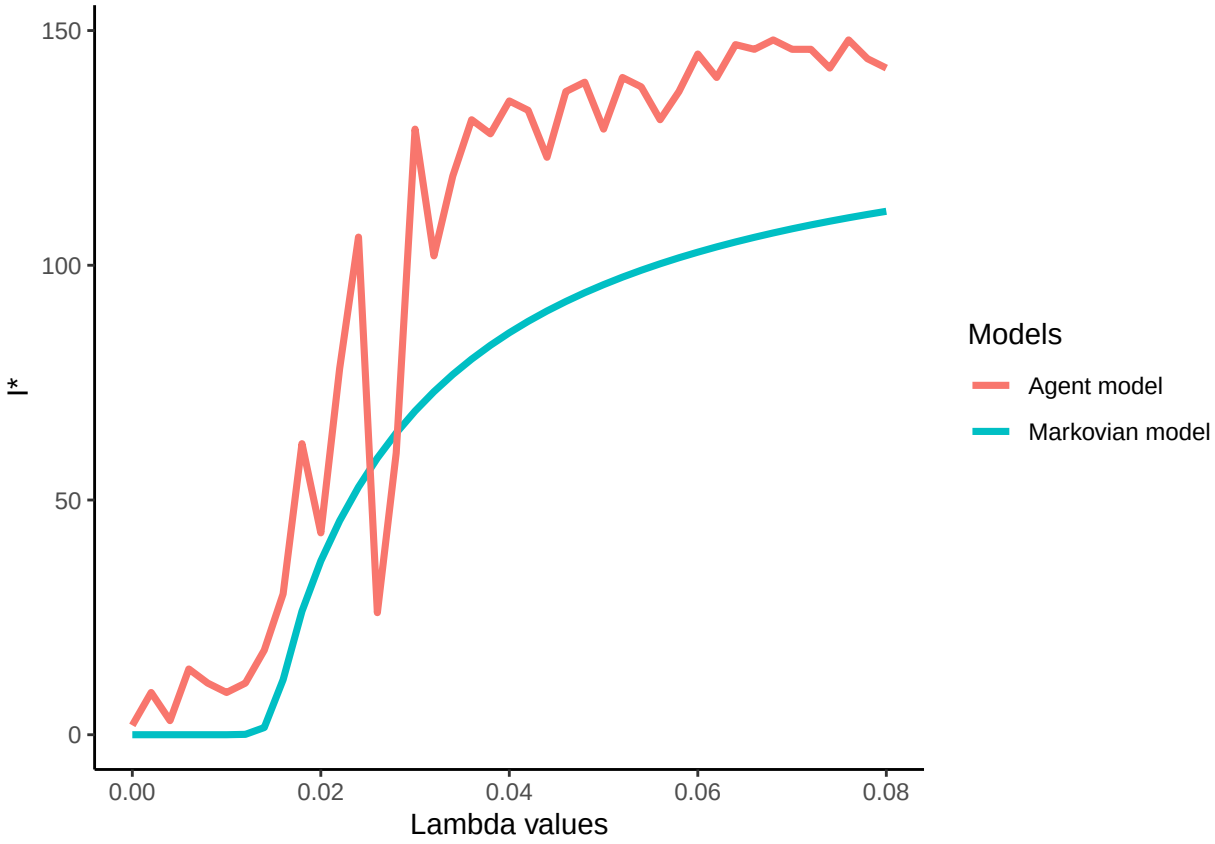
```
lambdas<-seq(from=0,to=0.08,by =0.002)
mus<-rep(0.15,length(lambdas))
Istable <- rep(NA,length(lambdas))

for(t in 1:length(lambdas)){
  Tsim <- 150
  lambda <- lambdas[t] #contagion rate
  mu <- mus[t] #recovery rate

  #Lists to store the values for each t from the infected and susceptible
  susceptible <- rep(NA,Tsim)
  infected <- rep(NA,Tsim)
  ids_nodes <-1:150

  #P_i(t) and q_i(t)
  q_i <- rep(0,150)

  #Initial conditions for P_i(0)
  P_i <- rep(3/150,150)

  for(j in 1:Tsim){
  #Storing the infection percentages
  infected[j] <- sum(P_i)/150
```

```
  susceptible[j] <- 1-infected[j]

  for(i in 1:150){
    #Updating q_i(t)
    neighbours <- ids_nodes[A1[i,]==1]
    if(length(neighbours)>0){
      q_i[i]<-1
      for(j in 1:length(neighbours)){
        q_i[i]<-q_i[i]*(1-lambda*P_i[j])
      }
      q_i[i]<-1-q_i[i]
    }
  }
  #Updating P_i(t)
  P_i <- P_i*(1-mu)+(1-P_i)*q_i
  }
  Istable[t]<-infected[length(infected)]*150
}

output<-data.frame(lambdas,Istable)

ggplot(data = output,aes(x=lambdas,y=Istable))+
geom_line(size=2)+
theme_classic()+
labs(x='Lambda values',y='I*')
```



What differences and similarities could be observed in the behaviour of both approaches? Let's compare the graphs of both stationary infected values to see their similarities (keeping in mind that they start from the same initial conditions).

```
output$Istable2<-I_graph
ggplot(data = output)+
geom_line(aes(x=lambdas,y=Istable,color = "Markovian model"),size=1.25)+
geom_line(aes(x=lambdas,y=Istable2,color = "Agent model"),size=1.25)+
theme_classic()+
labs(x='Lambda values',y='I*',color = "Models")
```



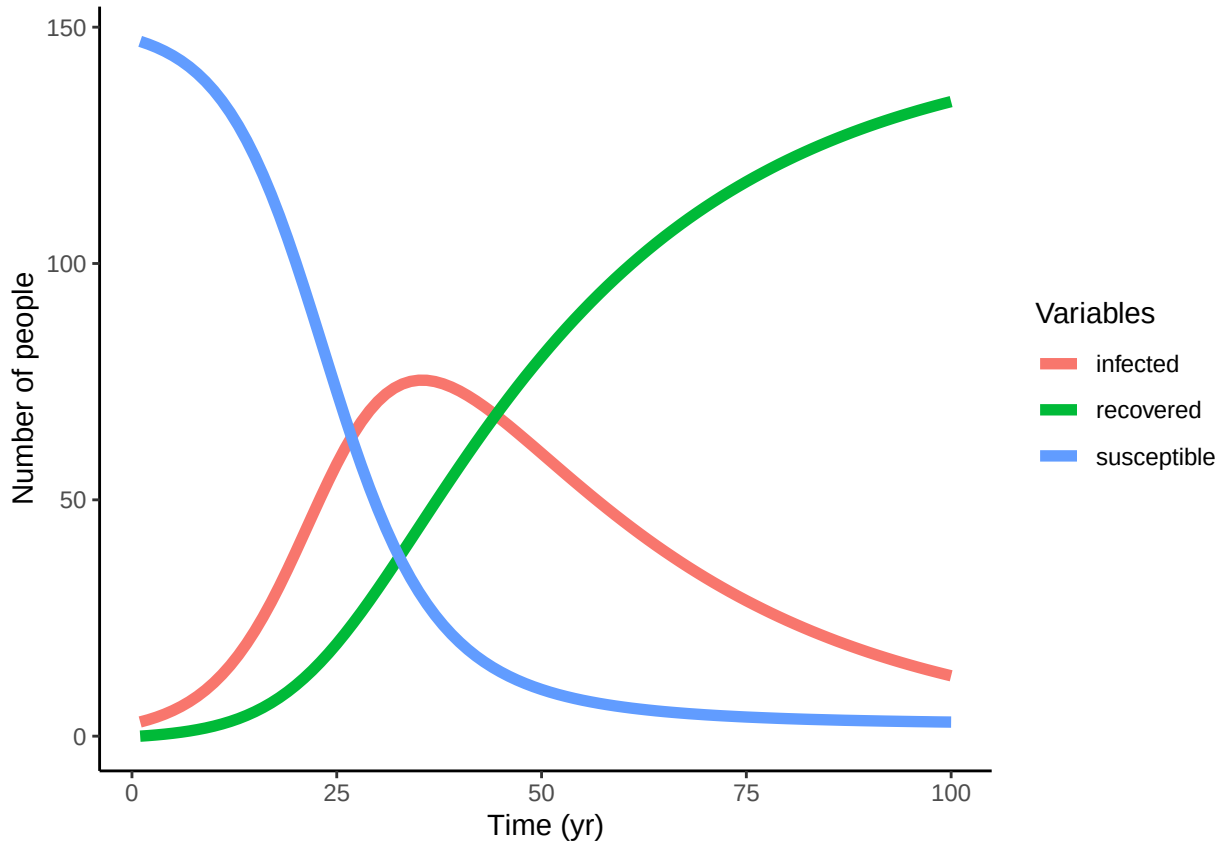
Exercise: just as in the agent based approach, modify the previous Markov algorithms to obtain the plots for an SIR model

```
Tsim <- 100
lambda <- 0.02 #Contagion rate
mu <- 0.035 #Recovery rate

#Lists to store the values for each t from the infected and susceptible
susceptible <- rep(NA,Tsim)
infected <- rep(NA,Tsim)
recovered <- rep(NA,Tsim)

ids_nodes <-1:150

#P_i(t) and q_i(t)
```

```
q_i <- rep(0,150)

r_i  <- rep(0,150)

#Initial conditions for P_i(0)
P_i <- rep(3/150,150)

for(t in 1:Tsim){
  #Storing the infection percentages
  infected[t] <- sum(P_i)/150
  recovered[t] <- sum(r_i)/150
  susceptible[t] <- 1-infected[t]-recovered[t]

  for(i in 1:150){
    #Updating q_i(t)
    neighbours <- ids_nodes[A1[i,]==1]
    if(length(neighbours)>0){
      q_i[i]<-1
      for(j in 1:length(neighbours)){
        q_i[i]<-q_i[i]*(1-lambda*P_i[j])
      }
      q_i[i]<-1-q_i[i]
    }
  }
  #Updating P_i(t)
  P_i <- P_i*(1-mu)+(1-P_i-r_i)*q_i
  r_i <- r_i + P_i*mu
}

rm(P_i,q_i,mu,lambda,i,ids_nodes,t,Tsim,neighbours,j)
```

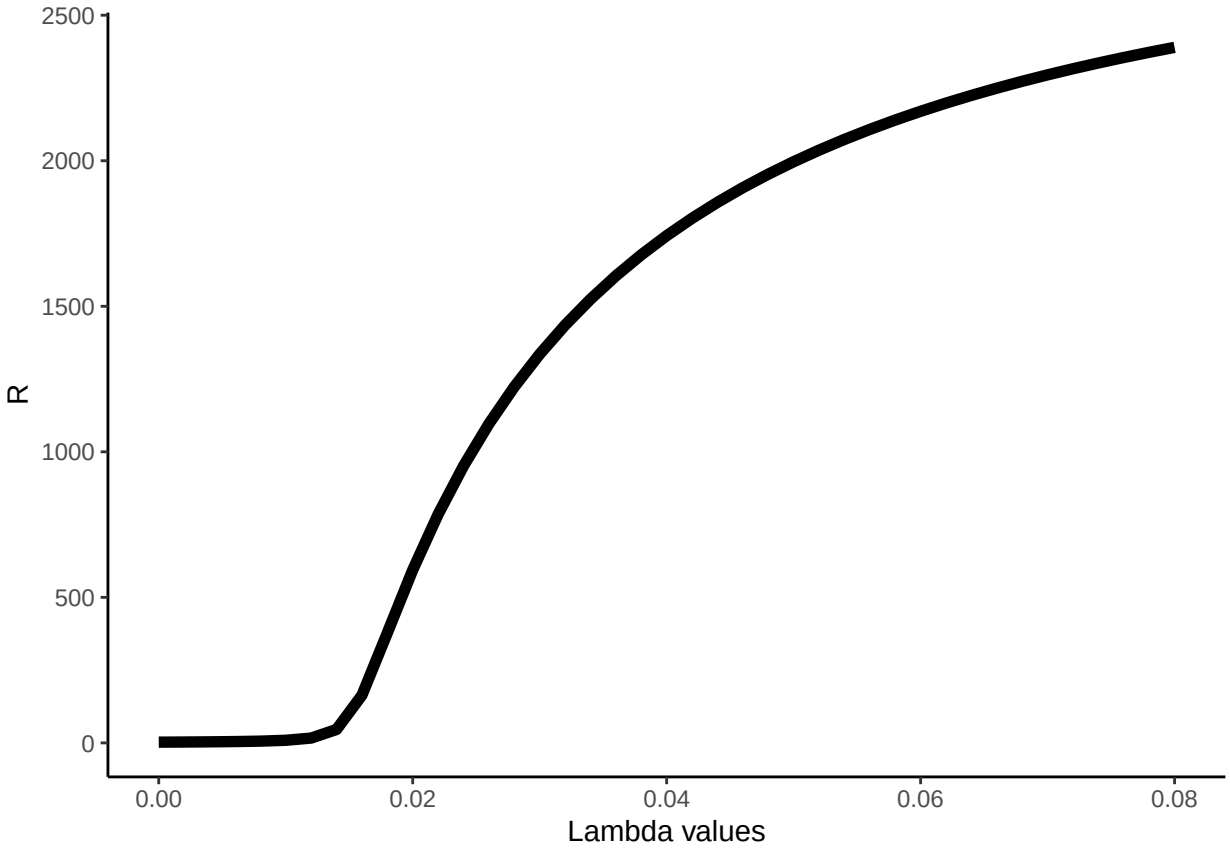
Let's see the behaviour of the curves:

```
output <-data.frame(time = 1:length(susceptible),susceptible = susceptible*150,infected = infected*150,
q<-output %>%
  gather(variable,value,-time) %>%
  ggplot(aes(x=time,y=value,color=variable))+
  geom_line(linewidth=2)+
  theme_classic()+
  labs(x='Time (yr)',y='Number of people',color = "Variables")
q
```




```
lambdas<-seq(from=0,to=0.08,by =0.002)
mus<-rep(0.15,length(lambdas))
Istable <- rep(NA,length(lambdas))

for(t in 1:length(lambdas)){
  Tsim <- 150
  lambda <- lambdas[t] #contagion rate
  mu <- mus[t] #recovery rate

  #Lists to store the values for each t from the infected and susceptible
  susceptible <- rep(NA,Tsim)
  infected <- rep(NA,Tsim)
  recovered <- rep(NA,Tsim)
  ids_nodes <-1:150

  #P_i(t) and q_i(t)
  q_i <- rep(0,150)
  r_i  <- rep(0,150)

  #Initial conditions for P_i(0)
  P_i <- rep(3/150,150)

  for(j in 1:Tsim){
  #Storing the infection percentages
  infected[j] <- sum(P_i)/150
  recovered[j] <- sum(r_i)/150
```

```
  susceptible[j] <- 1-infected[j]-recovered[j]

  for(i in 1:150){
    #Updating q_i(t)
    neighbours <- ids_nodes[A1[i,]==1]
    if(length(neighbours)>0){
      q_i[i]<-1
      for(j in 1:length(neighbours)){
        q_i[i]<-q_i[i]*(1-lambda*P_i[j])
      }
      q_i[i]<-1-q_i[i]
    }
  }
  #Updating P_i(t)
  P_i <- P_i*(1-mu)+(1-P_i)*q_i
  r_i <- r_i+P_i*mu
  }
  Istable[t]<-recovered[length(recovered)]*150
}

output<-data.frame(lambdas,Istable)

ggplot(data = output,aes(x=lambdas,y=Istable))+
geom_line(size=2)+
theme_classic()+
labs(x='Lambda values',y='R')
```


2500
2000
1500
1000
500
0
R
0.00
0.02
0.04
0.06
0.08
Lambda values