R programming

José R. Valverde
jrvalverde@cnb.csic.es
CNB/CSIC

Epidemiological models

Are statisticians normal?

What is this about?

- In this session we are **not** learning how to analyze epidemiology data
- We **will** learn how to simulate a basic epidemic model using R.
  - The **SIR** model for the spread of disease.

SIR

- In the SIR model we simulate spread of an epidemic disease that propagates from sick to healthy people. Infected people develops the disease and eventually recover.
- This gives us three possible kinds of person
  - **S**usceptible (healthy people who may acquire the disease)
  - **I**nfected people
  - **R**econvered patients

Compartmental models

- The SIR model is a specific type of **compatmental** model:
  - The population is classified into compartments, and people may “move” from one compartment to the other.
  - Suscpetible people may be infected
  - Infected people may recover from disease

Epidemics

- Disease spreads through the population
- Epidemics are sudden outbreaks of disease in a population where it was not present
- Endemics are diseases that never leave the population
- Understanding which factors make a disease suddenly spread and develop into an epidemy, ad how it may stabilize and become endemic is a major medical and biotechnological concern.

How does a disease spread?

- There are many mechanisms, some of them more efficient than others
- The number of persons that can be infected (on average) by an infected person is called the basic reproduction number. As a rule
  - If less than 1, the disease will die out
  - If greater than 1, the disease will spread and become an epidemic.

Building models

- We need to make a trade-off between simple models, which omit most details and show general qualitative behaviour, and detailed models, usually designed for specific situations and producing quantitative predictions
  - Detailed models are generally difficult
  - Simple models are inaccurate but they are
    - useful and
    - the basic building block for detailed models!

Dr. Ross was awarded
the second Nobel Prize in Medicine for his
demonstration of the dynamics of
the transmission of malaria between mosquitoes
and humans.

SIR revisited

- In the SIR model we simulate a disease that confers immunity against re-infection.
  - Recovered individuals cannot be re-infected

Other models

- SIS: susceptible → Infected → susceptible
  - The disease confers no immunity and recovered patients may be re-infected
- SEIR: susceptible → exposed → infected → recovered
  - You need to be exposed to the disease to be infected
- SEIS: susceptible → exposed → infected → susceptible
  - Same but no immunity
- SIRS: susceptible → infected → recovered → susceptible
  - Recovered patients are immune for some time but may lose immunity and become susceptible again (e.g. flu)

Modeling

- The basic method is the same in all cases
  - We divide the population into compartments and define a probability for individuals to transit from one compartment to the others.
  - Probability will be a function of time
  - TIME will be our independent variable (we may define others in more complex simulations).

The process

- We start with a number of individuals in each comparment and see how these numbers change over time.
- The probability of a transition from one compartment to another is a function of time
- We need to define these functions.

S I and R

- S: number of susceptible individuals
- I: number of infected indiviuals
- R: number of recovered individuals
- S_0, I_0, R_0 will be the initial numbers, which we will define to start our simulation
- S_t, I_t, R_t will be the number of individuals in each compartment at time t.

Calculating S, I and R

- We need to define how each of these values changes over time.
- The change will usually be a function of time **and** of the other values
  - The more infected persons, the easier it will be for a susceptible person to find an infected person and become infected as well
  - As one number changes the others must change as well (the total population remains constant)

Calculating changes

- We will calculate these dependences and define how each variable changes with time.
  - We shall compute the change per unit of time.
- The total population is constant

$$\frac{dS}{dt}+\frac{dI}{dt}+\frac{dR}{dt}=0$$

Susceptible

- The change in the number of susceptible people can only decrease (when they get infected)
  - As there is more susceptible people, there will be more people infected
  - As there are more infected people it will be more likely to get in contact with one and be infected
  - How much more likely? We'll adjust with a constant
    - $S' = dS/dt = - \beta S I$

Recovered

- This is easy: at each time t, a given proportion of people will recover, so
- $R' = dR/dt = \alpha I$

Infected

- The change in infected people has two components
  - the susceptible people that has become infected must be added to the number of infected people
  - Some people recovers, and are no longer infected, these must be substracted from the compartment

- $I' = dI/dt = \beta SI - \alpha I$

Define the three equations

- Create three R functions, one to define the change of each compartment in one unit of time.
- The only constant values are α and β
- dT is the unit of change of time and will be 1
  - So, we can leave it out
  - As long as we remember that we use 1 unit of time change! We will need to define what this unit is to scale the results.

examples

- deltaR ← function(alpha, I) alpha * I

- alpha = xxx (to be defined later)
- deltaR = function(I) alpha * I

- deltaT = 1
- deltaR = function(alpha, I, deltaT) alpha * I * deltaT

Make a SIR model

- Create a SIR function that computes the changes in t time points.
  - What we want is to specify the number of time points, the initial values of S, I and R, and the alpha and beta parameters.
  - Then we will compute new values for S, I and R at each time point using the equations

The function

- SIR <- function(S0, I0, R0, beta , alpha, t) { }
- We need to return the values of S, I and R at the t time points
  - We can use a data-frame composed of
    - A list of time points
    - Three lists with the computed S, I and R values

Preparing the lists

- We will need four vectors to store t_i, S_i, I_i, R_i
- These must contain t numerical values

```
S = numeric(t)
I = numeric(t)
R = numeric(t)
T = c(1:t)
# we already know the time values
```

Computing the values

- We already know the first value in each list

```
S[1] = S0
I[1] = I0
R[1] = R0
for (i in 2:t) {
    #Compute new values
}
```

Computing the values (2)

- $S_i = S_{i-1} + dS/dT$
- dS/dT is computed as the change from the last computed values

```
S[i] <- S[i - 1] - beta * S[i-1] * I[i-1]

R[i] <- R[i -1] + alpha * I[i-1]

I[i] <- I[I-1] + beta * S[i-1] * I[i-1] - alpha * I[i-1]
```

Returning the results

```
result <- data.frame(time=T,
                     S=S,
                     R=R,
                     I=I)
return (result)
```

A different solution

```
for (i in 2:t) {
    infected = beta * S[i-1] * I[i-1]
    recovered = alpha * I[i-1]
    S[i] = S[i-1] - infected
    R[i] = R[i-1] + recovered
    I[i] = I[i-1] + infected
                  - recovered
}
```

Another one

```
for (i in 1:t-1) {
    infected = beta * S[i] * I[i]
    recovered = alpha * I[i]
    S[i+1] = S[i] - infected
    R[i+1] = R[i] + recovered
    I[i+1] = I[i] + infected
                  - recovered
}
```

Being careful

- The computer does not work with infinite precision numbers
- It might happen that S < I at some point.
  - That would give us a negative value of S!
- The same may happen with I and R
- We can check it and stop the calculation

```
if (S <0 || I <0) break
```

Scaling

- It often makes sense to work with scaled variables. This simplifies calculations.
  - We may work with days, weeks, years...
  - But if we define the time unit as a day, week, year... the formula will always use 1
  - We will need to adapt the transition probability constants acoordingly
    - Flu lasts 2 weeks. If dT = 1 week, $\alpha = 0.5$. If dT = 1 day, then $\alpha = 1/14$, etc...

Try it

- For instance

```
SIR(S0 = 1-1e-6,
    I0 = 1e-6,
    R0 = 0.0,
    beta = 1.4247,
    alpha = 0.14286,
    t=70)
```

Plot it

- Assign the result to a variable
  - It is a data frame with vectors T, S, I and R
- Write a function to summarize the results:
  - Plot the three variables S, I and R versus time T
  - Add labels
  - Save it to a file

Doing it the “easy” way

- Package “deSolve” can be used to solve differential equations such as these.
- Function "ode" can be used to solve a system of ordinary differential equations:
  - Needs a function defining the system of equations
  - A vector with the initial values
  - A vector with any parameters needed
  - A vector of times to compute.

The system of equations

```
## Create an SIR function
sir <- function(time, compartemnts,
                parameters) {
    with(as.list(c(compartments, parameters)), {
        dS <- -beta * S * I
        dI <- beta * S * I - gamma * I
        dR <- gamma * I
        return(list(c(dS, dI, dR)))
    })
}
```

Doing it

```
initial_values <- c(S = 1-1e-6,
                    I = 1e-6,
                    R = 0.0)
parameters <- c(beta = 1.4247,
                gamma = 0.14286)
times <- seq(0, 70, by = 1)
out <- ode(y = initial_values, times = times,
       func = sir, parms = parameters)
out <- as.data.frame(out)
```

Yet another way

- If we code the probabilities of change from one compartment to each other, then we are actually defining a Markov Chain.
- Package “markovchain” allows us to use Markov Chains.
- We define the model as a matrix of transition probabilities

SIR matrix style

- An example matrix

→	S	I	R
S	0.9	0.1	0
I	0	0.8	0.2
R	0	0	1

SIR as Markov Chain

```
mcSIR <- new("markovchain",
      states=c("S","I","R"),
      TransitionMatrix =
        matrix(data=
        c(0.9,0.1,0,0,0.8,0.2,0,0,1),
        byrow=TRUE, nrow=3),
      name="SIR")
```

Showing SIR as MC

```
initialState <- c(99,0,1)

show(mcSIR)

plot(mcSIR, package="diagram")
```

Computing SIR as MC

```
timesteps <- 100
sir.df <- data.frame( "timestep" = numeric(),
   "S" = numeric(), "I" = numeric(),
   "R" = numeric(), stringsAsFactors=FALSE)

for (i in 0:timesteps) {
      newrow <-as.list(
               c(i,round(as.numeric(
                   initialState * mcSIR ^ i),0)))
      sir.df[nrow(sir.df) + 1, ] <- newrow
}
```

Lessons learned

- There are always many ways to solve a problem
- Different abstractions will lead you to each of them
- Choosing a good package makes work a lot easier
- You can do anything yourself
- Functions make your life easier
- Good variable names make your code easier to understand.

Thank You

Questions?