Stochastic compartmental models

This worksheet is adapted from a course module by Roland Regoes

Although conceptually the stochastic SIR model is more difficult than the deterministic one, it is not more difficult to simulate. This is due to the fact that the discrete variables are easier for computers to handle than continuous ones. To simulate the stochasticity of processes, however, requires the use of random number generators. Although random number generators are not straight-forward to develop on computers (which are inherently deterministic) almost every programming language has in-built ones nowadays.

The brute-force method to simulate our stochastic SIR model (with its continuous index space and discrete state space) would be to set very small time-steps in your simulation — so small that, at most, a single process happens at a given step. This small time step would be necessary to avoid problems arising from two or more processes happening at the same time step. The problem with this brute force approach is that in most time steps nothing will happen. This is not efficient. Fortunately, there are much cleverer algorithms. The following subsection introduces the so-called *Gillespie algorithm* which is a very efficient, but still accurate way to simulate stochastic models.

The deterministic SIR model with death

Here is a variant of the SIR model, which includes births and deaths:

$$dS\frac{d}{t} = m(S + I + R) - mS - bSI$$

$$dI\frac{d}{t} = bSI - (m + v)I - \gamma I$$

$$dR\frac{d}{t} = \gamma I - mR$$

The parameters are:

m host death rate

b infection rate

v pathogen-induced mortality

γ rate of recovery

The term that describes the birth of susceptible hosts, $m(S + I + R)$, ensures that deaths due to non-pathogen-related causes are balanced, and the total population $(S + I + R)$ would remain constant over time if there were no death due to the epidemic (expressed by vI).

The Stochastic SIR model

In the stochastic version of the SIR model, the continuous variables are replaced by discrete numbers, and the process rates are replaced by process probabilities. Let us denote the probability of the ith process by a_i; a will thus be a vector holding the probabilities of all possible processes. There are six such processes in our stochastic SIR models,

which are listed in Table 1. For example, at time t, the probability that a new susceptible host is infected is: $P(Infection) = bSI = a_5$

Process	**Probability**
Host birth	$a_1 = m(S + I + R)$
Death of susceptible host	$a_2 = mS$
Death of infected host	$a_3 = (m + v)I$
Death of recovered host	$a_4 = mR$
Infection	$a_5 = bSI$
Recovery	$a_6 = rI$

Table 1: Processes in the stochastic SIR model.

Figure 1:

The Gillespie algorithm

The idea of the Gillespie algorithm is that one first determines **when** something happens next. Suppose the current time is t. The time $t + \tau$ at which something happens next is an exponentially distributed random number scaled by the sum of all process rates, $\Sigma_i a_i$

```
tau <- rexp(1, sum(a))
```

> Exercise a: Look up the help documentation for rexp. What are the two arguments?

Then, the Gillespie algorithm determines **what** happens next. This is done by drawing a process randomly from all possible processes

according to their respective probabilities. This can be done easily in R by drawing the index of the next process with a weighted sample function:

```
sample(length(a),1,prob=a)
```

When we have determined which process happens, we can update the variables (the so-called state of the system). Then we iterate this process as long as we want.

> Exercise b: Look up the help documentation for sample. Then, use it to shuffle a vector with numbers from 1 to 10, and to randomly select one number from that vector

Rudimentary R-script

We supplied a starting script with a skeleton of a Gillespie algorithm for the stochastic SIR model. Please fill in the missing commands and try to make it work:

```
# set parameters
parms=c(m=1e-4,b=0.02,v=0.1,r=0.3)
initial=c(S=50, I=1, R=0)
time.window=c(0, 100)
# initialize state and time variables and
#write them into output
state <- initial
time <- time.window[1]
# define output dataframe
```

```
output <- data.frame(t=time,
S=state["S"], I=state["I"], R=state["R"],row.names=1)
# define how state variables S, I and
#R change for each process
processes <- matrix(0, nrow=6, ncol=3,
                        dimnames=list(c("birth",
                                        "death.S",
                                        "infection",
                                        "death.I",
                                        "recovery",
                                        "death.R"),
                                        c("dS","dI","dR")))
# process probabilities
probabilities <- function(state){
<...>
}
while(time < time.window[2] & state["I"]>0){
    # calculate process probabilities for current state
    <...>
    # WHEN does the next process happen?
    <...>
    # update time
    <...>
    # WHICH process happens after tau?
    <...>
    # update states
    <...>
    # write into output
    output <- rbind(output,c(time,state))
}
output
```

```
##Plot it!
```

Once you have completed this script and it works you should plot a few epidemics. Later on, when you are working on the exercises you will want to run the algorithm repeatedly with different parameters. An efficient way to do this is to write a function that includes the algorithm, and then you only need to call this function with the appropriate parameters from the main body of your program.

Here is an example of the kind of plot your code should generate: