Continuous-time stochastic simulation of epidemics in R

Ben Bolker

May 16, 2005

1 Introduction/basic code

Using the *Gillespie algorithm*, which assumes that all the possible events that can occur (death of an individual, birth, infection, etc.) occur *independently* and (given the current state of the system — number of infectives, susceptibles, etc.) *with constant probability per unit time*. Given that this is true, we can pick an exponential deviate (using R's `rexp()` function) that tells us how long it is until the next event occurs, and then pick from among the possible transitions with probabilities proportional to their individual rates (`sample()`).

Set the random-number seed so that results are reproducible every time we run this code (you can really use any integer you want here).

```
> set.seed(1001)
```

An R function that implements the Gillespie algorithm. The user must specify

- `start`: starting values
- `ratefun`: a function of the current state variables, parameters, and time that returns a numeric vector of the rates (probabilities per unit time) at which each kind of event is occurring
- `trans`: a matrix indicating the changes in each state variable (column) that occur when a particular event (row) takes place
- `pars`: a (named) numeric vector of parametesr
- `times`: a vector of times at which to report output

2 Simple SIR

Defining the appropriate inputs for a simple SIR model with no vital dynamics (nor any other complications).

Defining names for the state variables and transitions isn't absolutely necessary, but will make your code *much* easier to read later on:n

```
> statenames.SIR <- c("S", "I", "R")
> transnames.SIR <- c("infection", "death", "recovery")
```

Define the matrix of transitions (not the same as the transition matrix of a Markov chain) and the function :

The transition matrix ends up looking like this:

```
> trans.SIR

           S  I R
infection -1  1 0
death      0 -1 0
recovery   0 -1 1
```

Define parameters (numeric vector with names):

```
> pars.SIR <- c(beta = 0.1, alpha = 1, gamma = 1)
```

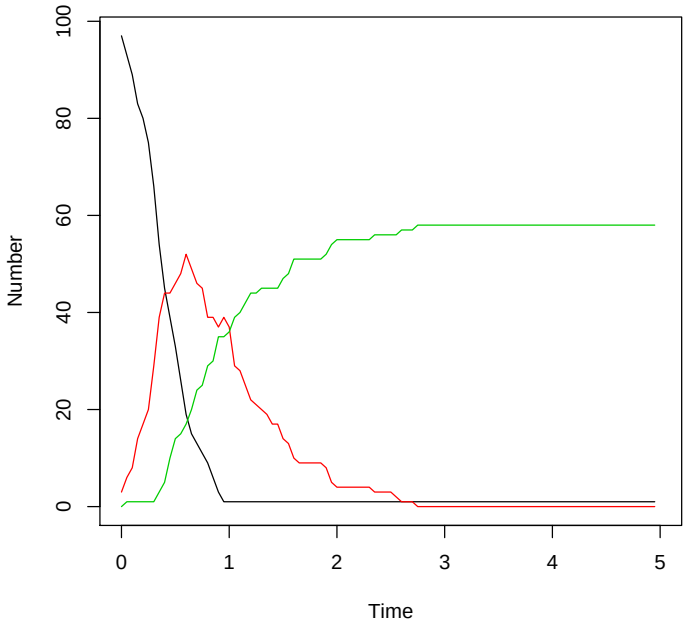
(R_0 in this case is equal to $\beta N/(\alpha + \gamma)$; if $N = 100$ then $R_0 = 5$).

Run stochastic simulation:

```
> G.SIR <- gillesp(start = c(S = 97, I = 3, R = 0), times = seq(0,
+     5, by = 0.05), ratefun = ratefun.SIR, trans = trans.SIR,
+     pars = pars.SIR)
```

Plot it (`matplot` plots the columns of a matrix as separate lines against a single x variable. `type="l"` specifies lines, `lty=1` specifies solid lines, `xlab` and `ylab` specify x- and y-axis labels. `G.SIR[,"times"]` picks out the column labeled `times`, `G.SIR[,-1]` picks out all but the first column).

```
> matplot(x = G.SIR[, "times"], y = G.SIR[, -1], type = "l", lty = 1,
+     xlab = "Time", ylab = "Number")
```

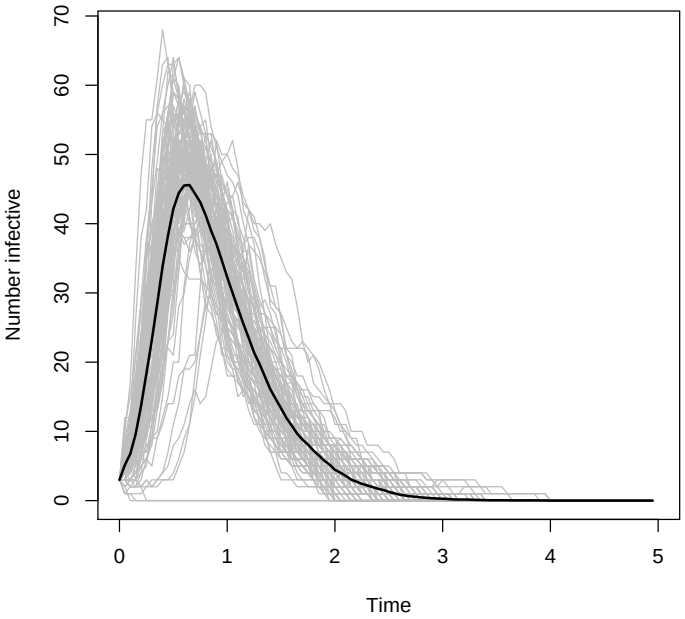


Fancy: use `replicate` run 100 stochastic simulations, using `[,"I"]` to save just the number of infectives ...

```
> G.SIR.mult <- replicate(100, gillesp(start = c(S = 100, I = 3,
+     R = 0), times = seq(0, 5, by = 0.05), ratefun = ratefun.SIR,
+     trans = trans.SIR, pars = pars.SIR)[, "I"])
```

Plot it, adding a line for the mean (`lines` adds a line to the existing plot, `lwd=2` sets the line width to 2):

```
> matplot(G.SIR[, "times"], G.SIR.mult, type = "l", col = "gray",
+     lty = 1, xlab = "Time", ylab = "Number infective")
> lines(G.SIR[, "times"], rowMeans(G.SIR.mult), lwd = 2)
```



3 SIR with vital dynamics (constant population size)

```
> ratefun.vSIR <- function(X, pars, time) {
+     vals <- c(as.list(pars), as.list(X))
+     rates <- with(vals, c(birth = mu * K, infection = beta *
+         S * I, Sdeath = (mu * S), Ideath = (alpha + mu) * I,
+         Rdeath = (mu * R), recovery = gamma * I))
+ }
> statenames.vSIR <- statenames.SIR
> transnames.vSIR <- c("birth", "infection", "Sdeath", "Ideath",
+     "Rdeath", "recovery")
> trans.vSIR <- matrix(c(1, 0, 0, -1, 1, 0, -1, 0, 0, 0, -1, 0,
+     0, 0, -1, 0, -1, 1), byrow = TRUE, ncol = 3, dimnames = list(transnames.vSIR,
+     statenames.vSIR))
```

Here I'm going to make β quite a bit larger ($R_0 = \beta N/(\alpha+\gamma+\mu) = 150/2.2 = 68$) (!), in order to avoid having the epidemic go extinct during the crash after the first epidemic peak. There is a general epidemiological puzzle here about how new epidemics get started from low numbers without going extinct in the first epidemic trough ...

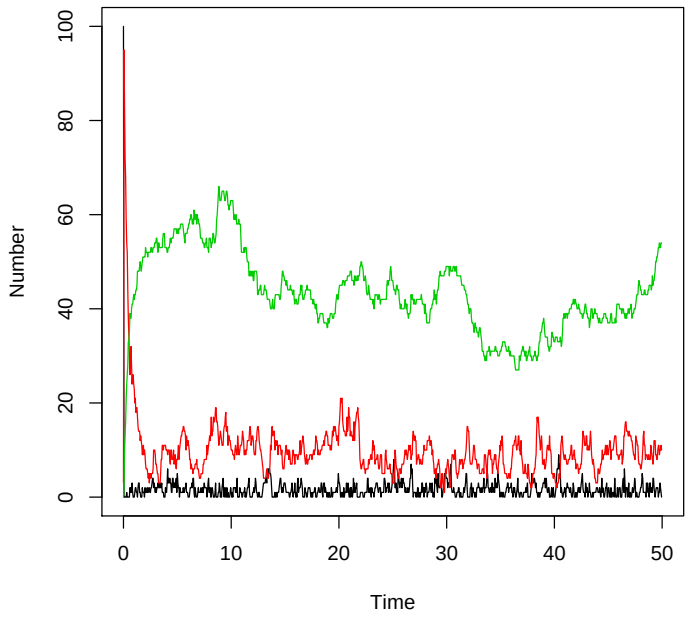
```
> pars.vSIR <- c(beta = 1.5, alpha = 1, gamma = 1, mu = 0.2, K = 100)
```

Run it:

```
> G.vSIR <- gillesp(start = c(S = 100, I = 3, R = 0), times = seq(0,
+     50, by = 0.05), ratefun = ratefun.vSIR, trans = trans.vSIR,
+     pars = pars.vSIR)
```

Plot it:

```
> matplot(G.vSIR[, "times"], G.vSIR[, -1], type = "l", lty = 1,
+     xlab = "Time", ylab = "Number")
```



4 Extensions

- vertical transmission: split `birth` into `Sbirth` and `Ibirth`
- vaccination: e.g.

  ```
  > rates <- c(Sbirth = ifelse(time < vaccstart, b * N, (1 - vaccrate) *
  +     b * N), Rbirth = ifelse(time < vaccstart, 0, vaccrate * b *
  +     N))
  ```

- waning immunity

- multiple infected classes to simulate gamma/non-exponential infectious periods
- exposed class
- density-dependent birth/death rates
- density-dependent disease-induced mortality???
- seasonal variation in rates: e.g. sinusoidal

  ```
  > beta.seas <- beta.0 * (1 + beta.ampl * cos(2 * pi * time))
  ```

  or step-function:

  ```
  > beta <- ifelse(time%%1 < seas.frac, beta.0, 0)
  ```

- multiple classes of individuals with differential mixing/susceptibility/etc. (S1,I1,R1,S2,I2,R2, etc., although this can get tedious)

5 Other challenges

- Write `odesolve`/`lsoda` code to check against these results ...