Simulating genealogies with an epidemiological coalescent model using `rcolgem`

Erik M Volz

February 10, 2016

This vignette will demonstrate how to build models with rcolgem and simulate genealogies using the structured coalescent. We will use simple deterministic and stochastic HIV models where infected hosts progress through several stages of infection characterised by different transmission rates.

First consider a very simple model of an HIV epidemic using ordinary differential equations. In this model, the infectious period will be broken into three stages of different average duration and with different transmission rates. The first stage, *early HIV infection* (EHI) is short (average duration $1/\gamma_0$=1 year), but has high transmission rate β_0. The second stage, *chronic HIV infection*, is long (average duration $1/\gamma_1$=7 years), and has small transmission rate. The last stage, AIDS, lasts $1/\gamma_2 = 2$ years on average and has an intermediate transmission rate. There are births into the susceptible state at rate bN where $N = S+I_0+I_1+I_2$. And there is mortality due to natural causes from all states at rate μ. The parameter values are listed in table 1. The model equations are as follows:

$$\dot{S} = bN - \mu S - (\beta_0 I_0 + \beta_1 I_1 + \beta_2 I_2)S/N \quad (1)$$

$$\dot{I_0} = (\beta_0 I_0 + \beta_1 I_1 + \beta_2 I_2)S/N - (\mu + \gamma_0)I_0 \quad (2)$$

$$\dot{I_1} = \gamma_0 I_0 - (\mu + \gamma_1)I_1 \quad (3)$$

$$\dot{I_2} = \gamma_1 I_1 - (\mu + \gamma_2)I_2 \quad (4)$$

The model is also illustrated in figure 1.

The package is loaded by

```
> library(rcolgem)
```

Now we need to build this model in a format that can be understood by the package and used to simulate trees. To do this, we will use the `build.demographic.process` function. We first need to express the equations in a canonical format. According to this format, we will tally birth and migration events between demes. In our example, the deme corresponds to the stage of infection that an infected host can be in, so we will refer the demes with the following names:

```
> INFECTEDNAMES <- c('I0', 'I1', 'I2')
```

Table 1: Parameter symbols and values.

Parameter	Symbol	Value
Duration EHI	$1/\gamma_0$	1 year
Duration chronic	$1/\gamma_1$	7 years
Duration AIDS	$1/\gamma_2$	2 years
Birth rate	b	0.036
Natural death rate	μ	1/30
EHI transmission rate	β_0	1.2
Chronic transmission rate	β_1	0.03
AIDS transmission rate	β_2	0.09
Initial no. susceptibles	$S(0)$	3000

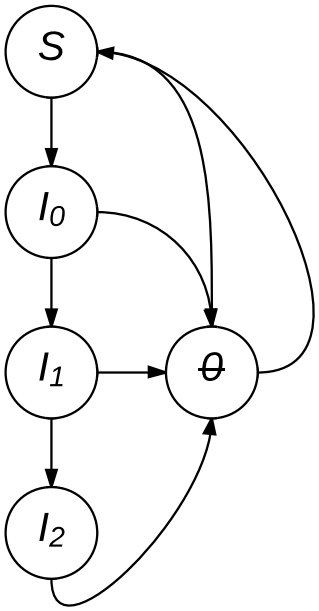


Figure 1: An illustration of the HIV compartmental model.

There are $m = 3$ demes in this model, so the birth events between demes needs to be expressed with a 3×3 matrix F. The element F_{kl} represents the rate of transmissions by a host in deme k to a host in deme l. In our example, this is the following:

```
> births <- rbind(
+   c('beta0 * S * I0 / (S + I0 + I1 + I2)', '0', '0'),
+   c('beta1 * S * I1 / (S + I0 + I1 + I2)', '0', '0'),
+   c('beta2 * S * I2 / (S + I0 + I1 + I2)', '0', '0')
+ )
> rownames(births)=colnames(births)<- INFECTEDNAMES
```

Each element of the matrix is a string that will be parsed as C++ code and evaluated using the Rcpp package, so it is important to write it exactly as you would if you were solving the equations with the C++. In this case, the parameters (beta, gamma etc) are automatically made available. This is the recommended way to write the model, since after compilation, simulation of the model will be very fast. Alternatively, we express the equations as R expressions, in which case the parameters will be available as a list called `parms`. For example:

```
> ## births <- rbind(
> ##   c('parms$beta0 * S * I0 / (S + I0 + I1 + I2)', '0', '0'),
> ##   c('parms$beta1 * S * I1 / (S + I0 + I1 + I2)', '0', '0'),
> ##   c('parms$beta2 * S * I2 / (S + I0 + I1 + I2)', '0', '0')
> ## )
```

Note that there are zero rates in the 2nd and third columns, since all new infected hosts start out in the first stage of infection (EHI). Also note that we *must* give row and column names to the matrix, and these names must correspond to the names of the demes.

Similarly, we must create a matrix of migrations:

```
> migrations <- rbind(
+   c('0', 'gamma0 * I0', '0'),
+   c('0', '0', 'gamma1 * I1'),
+   c('0', '0', '0')
+ )
> rownames(migrations)=colnames(migrations) <- INFECTEDNAMES
```

Note that this matrix tallys the stage progression from EHI to chronic and from chronic to AIDS.

We must also write a vector of expressions for events that terminate a lineage– In this model, this occurs due to natural or AIDS related mortality:

```
> deaths <- c(
+  'mu*I0'
+  , 'mu*I1'
+  , 'mu*I2 + gamma2 * I2'
+ )
> names(deaths) <- INFECTEDNAMES
```

Finally, we must write a vector of rates for state variables that do not correspond to demes in the coalescent model. In our example, there is only one such variable- the number of susceptibles:

```
> nonDemeDynamics <- c( S = '-mu*S + mu*(S + I0 + I1 + I2) -
+    S * (beta0*I0+beta1*I1+beta2*I2) / (S + I0 + I1 + I2)'
+ )
```

Note well that in all cases, the expression or equation must have the corresponding name of the state variable.

Now we can construct the demographic process:

```
> demo.model <- build.demographic.process(
+   births
+   , nonDemeDynamics
+   , migrations=migrations
+   , deaths=deaths
+   , parameterNames = c(
+      'beta0'
+      , 'beta1'
+      , 'beta2'
+      , 'gamma0'
+      , 'gamma1'
+      , 'gamma2'
+      , 'mu')
+   , rcpp = TRUE
+   , sde=TRUE
+ )

[1] "Wed Feb 10 19:26:25 2016 Compiling model..."
[1] "Wed Feb 10 19:26:43 2016 Model complete"

> class(demo.model)

[1] "demographic.process" "function"
```

This creates the model ('demo.model') by parsing the equations provided in births, migrations etc. Note also that we must provide the names of parameters that will be needed when compiling the model. Two options allow the user to customise the type of model that is created:

- rcpp : If TRUE, the equations are interpreted as C++ code and compiled using the Rcpp and inline packages. Compilation is slow, but simulating the model will be much faster. If FALSE, the equations are interpreted as R expressions. There is no pre-compilation step, but simulation will be much slower. This approach has some added flexibility, since non-scalar parameters (even other R functions) can be made available to the equations in the `parms` object.

- sde : If TRUE, the model will treat the equations as rates within a system of stochastic differential equations which are solved using the Euler method. If FALSE, the equations are treated as ODEs and solved using the deSolve package.

Now we can simulate the demographic process using

```
> ## demo.model(theta, x0, t0, t1, res = 1e3, integrationMethod='adams')
```

- theta is a named vector of all parameters
- x0 is a named vector of the initial conditions, eg I0, I1, I2 and S
- t0 and t1 are scalar times at which the process is initiated and terminated
- res provides the time resolution of the process, and also corresponds to the timestep if solving SDEs. Larger values will generally provide a more accurate approximation, but will be slower
- If solving ODEs, the integrationMethod parameter selects the method used by the deSolve package

Note well that an alternative to using the `build.demographic.process` function would be to manually construct the `demo.model` function. This may be a good alternative if the model is highly complex or to optimise simulation time. The return value of this function should be a list with four elements:

1. A length m vector giving the time of each simulated output
2. A list of length res; each element should be an $m \times m$ matrix with computed birth rates corresponding to each element in times
3. A list of length res; each element should be an $m \times m$ matrix with computed migration rates
4. A list of length res; each element should be a length m vector of population size within each deme

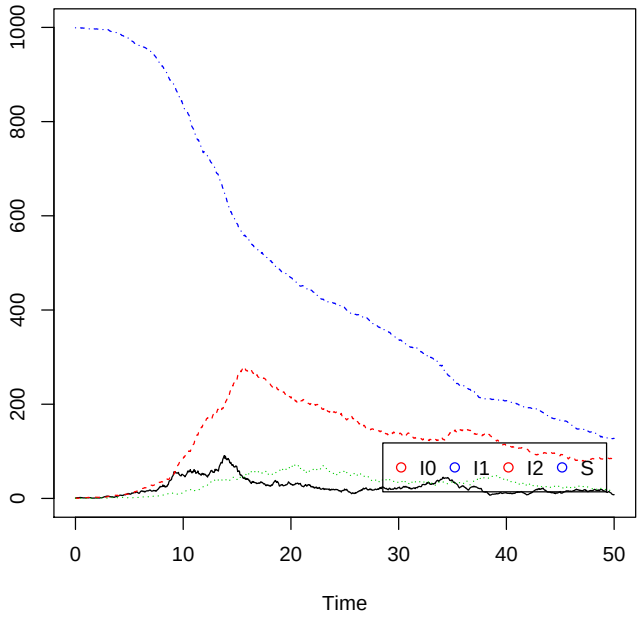
Let's pick some parameters and initial conditions:

```
> theta <- c(gamma0 = 1
+  , gamma1 = 1/7
+  , gamma2 = 1/2
+  , mu = 1/30
+  , beta0 = 12./10
+  , beta1=3./100
+  , beta2=9./100
+ )
> t0 <- 0
> t1 <- 50
> x0 <- c(S = 999, I0 = 1, I1 =.1, I2 = .1)
```

Note that the first stage of infection has a much higher transmission rate than subsequent stages.

We can easily visualise the model:

```
> show.demographic.process(demo.model, theta, x0, t0, t1)
```



Additional graphical parameters may be passed to matplot.

Now to simulate the tree, we must further specify the time that each lineage is sampled, and the state of the lineage at time of sampling. Let's create a vector of uniformly spaced sample times:

```
> n <- 100
> sampleTimes <- seq( 15, 25, length.out = n)
```

The states of the lineages are specified in the form of a $n \times m$ matrix, with element (i,j) corresponding to the probability that lineage i is in deme j when sampled. Let's construct such a matrix using multinomial sampling, so that most samples have chronic infection, and a few are sampled in the first stage:

```
> sampleStates <- t(rmultinom( n, size = 1, prob = c(.025, .9, .075) ))
> head(sampleStates)
```

```
     [,1] [,2] [,3]
[1,]    0    1    0
[2,]    0    1    0
```

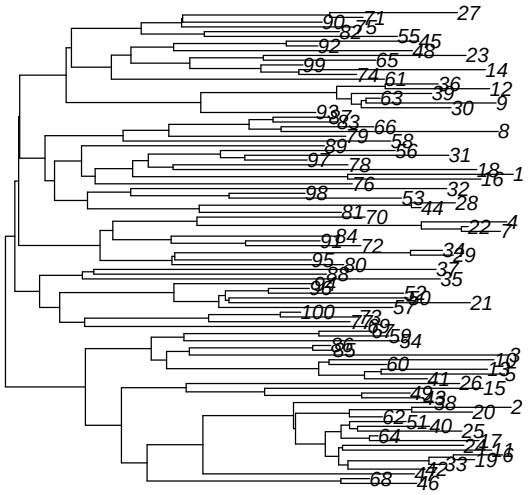
```
[3,]    0    1    0
[4,]    0    0    1
[5,]    0    1    0
[6,]    0    1    0
```

Now we can simulate the tree:

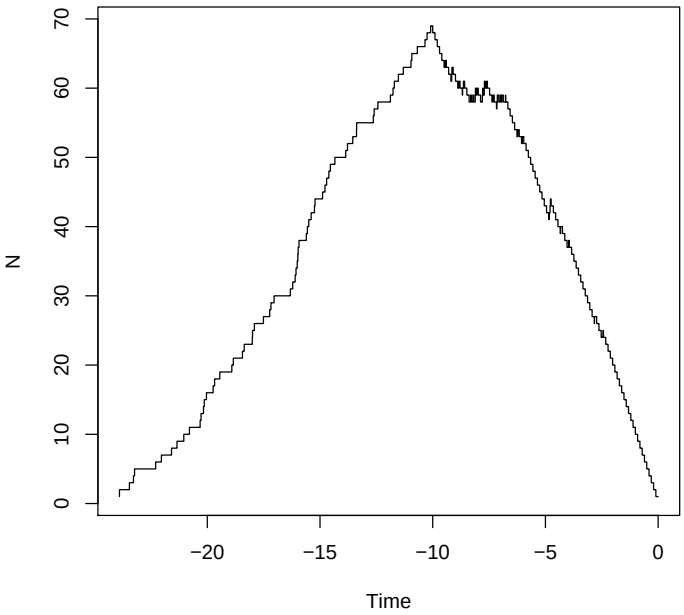
```
> tree <- sim.co.tree (theta, demo.model, x0, t0, sampleTimes, sampleStates, res = 1e3)
> tree

Phylogenetic tree with 100 tips and 99 internal nodes.

Tip labels:
        46, 68, 47, 42, 33, 19, ...

Rooted; includes branch lengths.
```

The return value is a 'DatedTree' object, which is derived from 'ape::phylo'. Consequently, all of the convenience functions for 'ape::phylo' also work:

```
> plot.phylo(tree)
```




```
> ltt.plot(tree)
```


N
0
10
20
30
40
50
60
70
−20
−15
−10
−5
0
Time