EpiModel Presentation

Owen R. Page

April 25, 2018

Contents

EpiModel is a package that can be used for studying mathematical modeling of infectious disease. This package supports deterministic models, stochastic individual contact models, and stochastic network models while also modeling for SI, SIR, and SIS systems with and without demography. Furthermore, this package provides the foundation to create a variety of biological models of both discrete and continual nature. All tutorial information was obtained from http://www.epimodel.org/tut.html and is in no way my own work (though I did alter the code from here to there).

First we shall begin with an overview of some functions, start with deterministic compartmental model (which means that the populations are divided into groups that represent very specific disease states) models, and work our way towards stochastic models. In DCMS, time is discrete, population is treated as a whole, and parameters are specified.

DCMs To simulate using EpiModel, we use the `dcm` function. Prior to running `dcm()` we must use the three following functions : `parm.dcm()` in which the model parameters are entered in which the arguments `inf.prob` to set transmission probability per act and `act.rate` set actions per person per unit time, `init.dcm()` which collects initial conditions, and `control.dcm()` which collects other model controls like type and timesteps needed for simulation.

Basic SI model

We will begin by first discussing a Susceptible-Infected (SI) model with random interaction in a population in a closed population. In such a model once an individual has become infected they remain infected.The size of the compartments is represented by the equations:

dS/dt= -lambda$S$ dI/dt=$lambda$S where lambda indicates the force of infection and lambda =Beta$c$I/N where Beta is the probability of transmission per contact, c is rate of contact per person/time, I is number of infected at time t, and N is population at time t.

```
#prelim:
library(EpiModel)
```

```
## Warning: package 'EpiModel' was built under R version 3.4.4
```

```
## Loading required package: deSolve
```

```
## Loading required package: ggplot2
```

```
## Loading required package: networkDynamic
```

```
## Loading required package: network
```

```
## network: Classes for Relational Data
## Version 1.13.0 created on 2015-08-31.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##                     Mark S. Handcock, University of California -- Los Angeles
##                     David R. Hunter, Penn State University
##                     Martina Morris, University of Washington
##                     Skye Bender-deMoll, University of Washington
##  For citation information, type citation("network").
##  Type help("network-package") to get started.
```

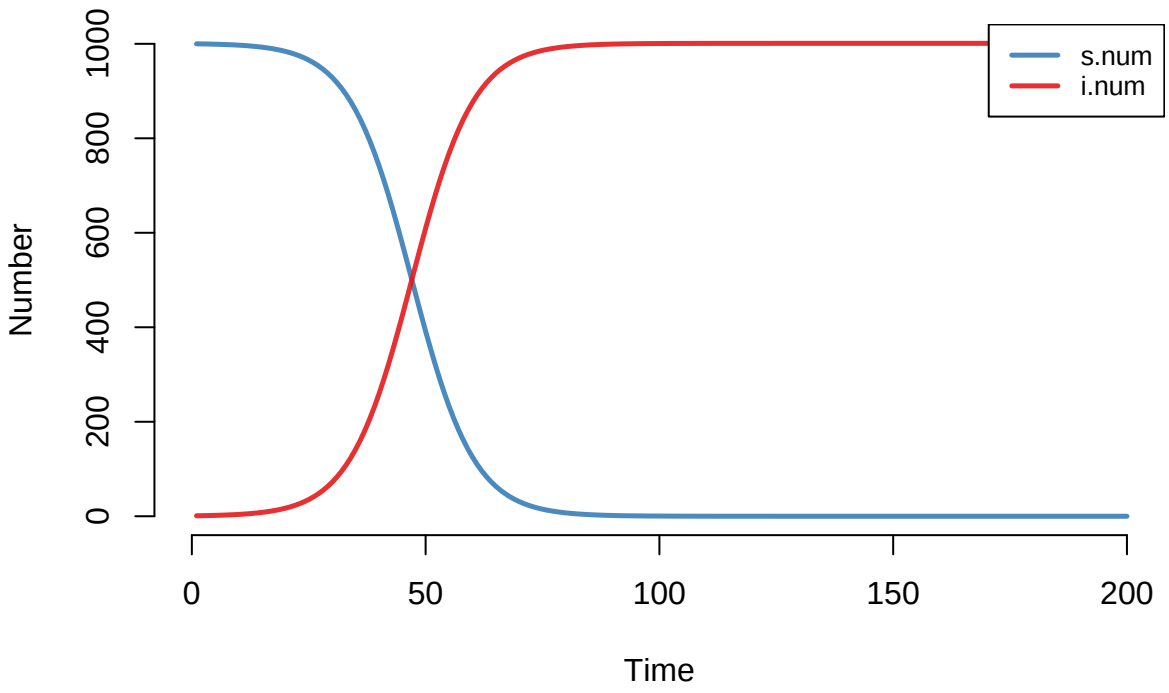
```
##
## networkDynamic: version 0.9.0, created on 2016-01-12
## Copyright (c) 2016, Carter T. Butts, University of California -- Irvine
##                     Ayn Leslie-Cook, University of Washington
##                     Pavel N. Krivitsky, University of Wollongong
##                     Skye Bender-deMoll, University of Washington
##                     with contributions from
##                     Zack Almquist, University of California -- Irvine
##                     David R. Hunter, Penn State University
##                     Li Wang
##                     Kirk Li, University of Washington
##                     Steven M. Goodreau, University of Washington
##                     Jeffrey Horner
##                     Martina Morris, University of Washington
## Based on "statnet" project software (statnet.org).
## For license and citation information see statnet.org/attribution
## or type citation("networkDynamic").

## Loading required package: tergm

## Loading required package: statnet.common

##
## Attaching package: 'statnet.common'

## The following object is masked from 'package:base':
##
##     order

## Loading required package: ergm

##
## ergm: version 3.8.0, created on 2017-08-18
## Copyright (c) 2017, Mark S. Handcock, University of California -- Los Angeles
##                     David R. Hunter, Penn State University
##                     Carter T. Butts, University of California -- Irvine
##                     Steven M. Goodreau, University of Washington
##                     Pavel N. Krivitsky, University of Wollongong
##                     Martina Morris, University of Washington
##                     with contributions from
##                     Li Wang
##                     Kirk Li, University of Washington
##                     Skye Bender-deMoll, University of Washington
## Based on "statnet" project software (statnet.org).
## For license and citation information see statnet.org/attribution
## or type citation("ergm").

## NOTE: Versions before 3.6.1 had a bug in the implementation of the
## bd() constriant which distorted the sampled distribution somewhat.
## In addition, Sampson's Monks datasets had mislabeled vertices. See
## the NEWS and the documentation for more details.

##
## tergm: version 3.4.1, created on 2017-09-12
## Copyright (c) 2017, Pavel N. Krivitsky, University of Wollongong
##                     Mark S. Handcock, University of California -- Los Angeles
##                     with contributions from
##                     David R. Hunter, Penn State University
```

```
##                     Steven M. Goodreau, University of Washington
##                     Martina Morris, University of Washington
##                     Nicole Bohme Carnegie, New York University
##                     Carter T. Butts, University of California -- Irvine
##                     Ayn Leslie-Cook, University of Washington
##                     Skye Bender-deMoll
##                     Li Wang
##                     Kirk Li, University of Washington
## Based on "statnet" project software (statnet.org).
## For license and citation information see statnet.org/attribution
## or type citation("tergm").
```

```
param <- param.dcm(inf.prob = 0.3, act.rate = 0.5) # setting parameters using arguments, inf.prob is th
init <- init.dcm(s.num = 1000, i.num = 1) # set initial values
control <- control.dcm(type = "SI", nsteps = 200) # set type and timestep number

SImod <- dcm(param, init, control) # creation of model from starter functions
SImod # prints in console very accessible summary of model
```

```
## EpiModel Simulation
## =======================
## Model class: dcm
##
## Simulation Summary
## ----------------------
## Model type: SI
## No. runs: 1
## No. time steps: 200
## No. groups: 1
##
## Model Parameters
## ----------------------
## inf.prob = 0.3
## act.rate = 0.5
##
## Model Output
## ----------------------
## Variables: s.num i.num si.flow num
```

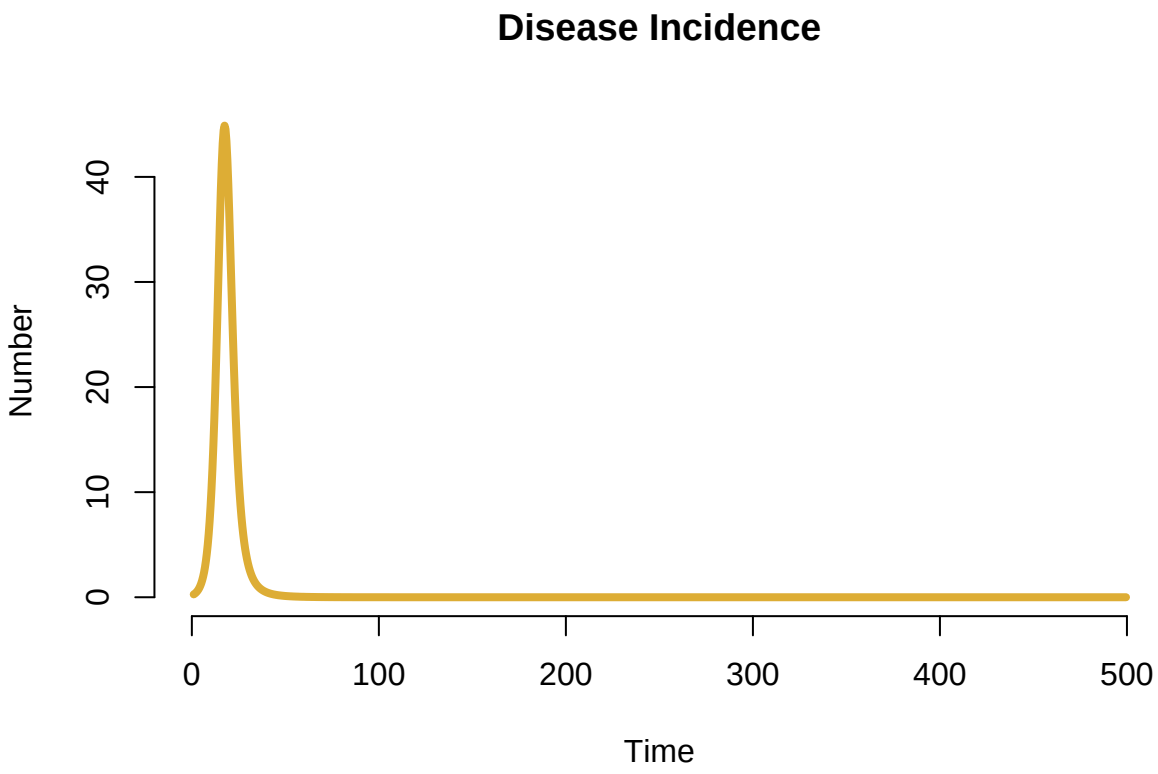
```
plot(SImod) # plot of basic SI model
```



SIR In an SI model, there is no possibility for recovery, but in an SIR model there is.Nu is the recovery rate which will represent the reciprocal of the average disease rate dS/dt = -lambda$S$ + N dI/dt = $lambda$S - nu$I$ dR/dt = nuI

First SIR without demography

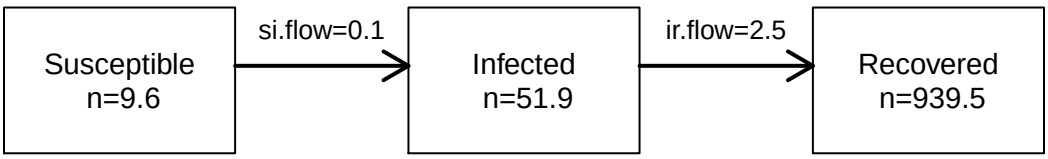
```
param <- param.dcm(inf.prob = 0.33, act.rate = 1.5, rec.rate = 1/10) # must include rec.rate
init <- init.dcm(s.num = 1000, i.num = 1, r.num = 0)  #adding in r compartment
control <- control.dcm(type = "SIR", nsteps = 500, dt = 0.5) # dt is here so we can obtain data within
SIRmod <- dcm(param, init, control)
plot(SIRmod, y = "si.flow", lwd = 4, col = "goldenrod",main = "Disease Incidence", legend = "n")
```



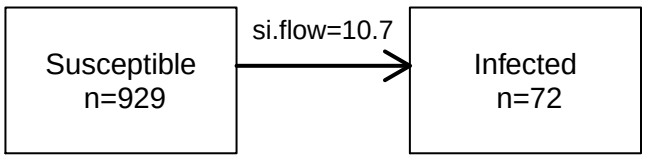

```
par(mfrow = c(1, 1))
#comp_plot(SIRmod_2, at = 30, digits = 1)
```

Flow digrams of previous models enable us to visualize the information via what is known as a stat-flow diagram:

```
#for SI
par(mfrow = c(1, 1))
comp_plot(SIRmod, at = 50, digits = 1)
```




```
#for SIR
par(mfrow = c(1, 1))
comp_plot(SImod, at = 30, digits = 1)
```



SIS modeling:

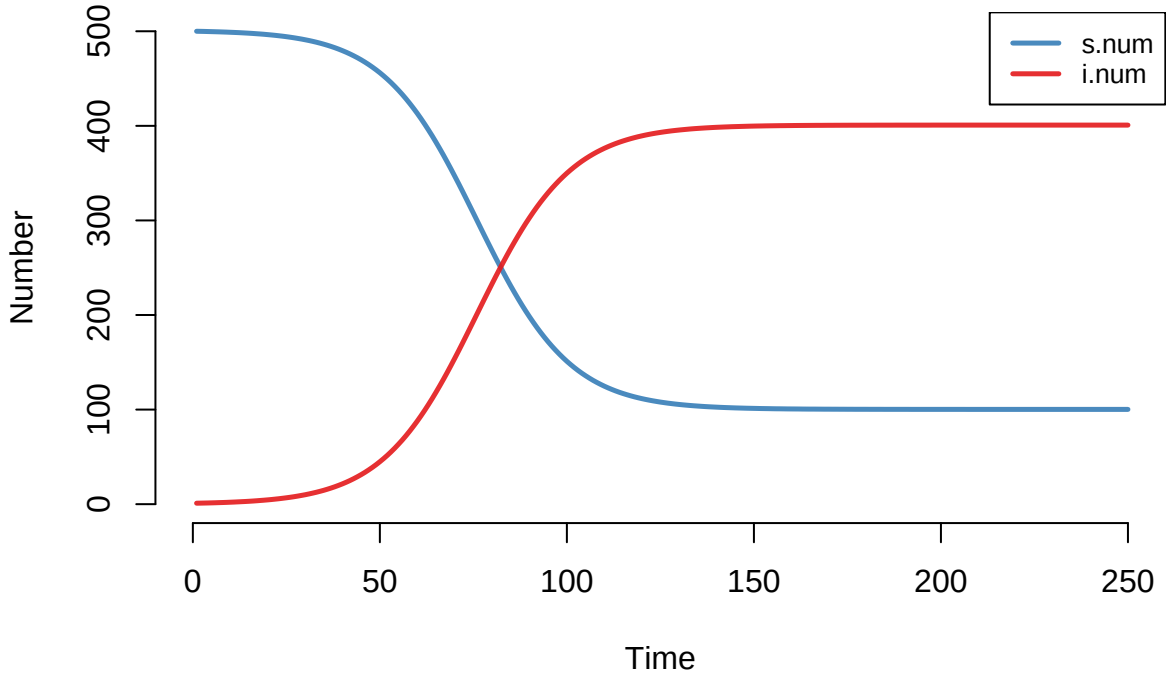
Next we will discuss another `dcm` known as an SIS model where there is really no life-long recovery. Note the following ODEs as representation: dS/dt = -lambda$S$ + nuI dS/dt = -lambda$S$ + nuI Note that nu now represents recovery but back into the susceptible compartment. We must specify the type of `dcm` by stating that `type="SIS"` and provide a `rec.rate`. This particular model will have a closed population.

```
param <- param.dcm(inf.prob = 0.2, act.rate =  0.5, rec.rate = 0.02)#vector here
init <- init.dcm(s.num = 500, i.num = 1)
control <- control.dcm(type = "SIS", nsteps = 250)
SISmod <- dcm(param, init, control)
SISmod
```

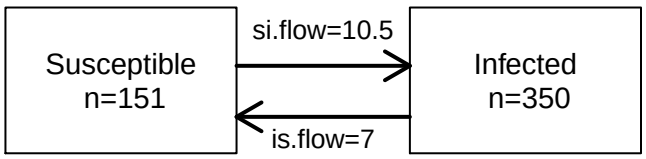
```
## EpiModel Simulation
## =======================
## Model class: dcm
##
## Simulation Summary
## ----------------------
## Model type: SIS
## No. runs: 1
## No. time steps: 250
## No. groups: 1
##
## Model Parameters
## ----------------------
## inf.prob = 0.2
## act.rate = 0.5
```

```
## rec.rate = 0.02
##
## Model Output
## -----------------------
## Variables: s.num i.num si.flow is.flow num
```

```
plot(SISmod) # to give a generic plot
```



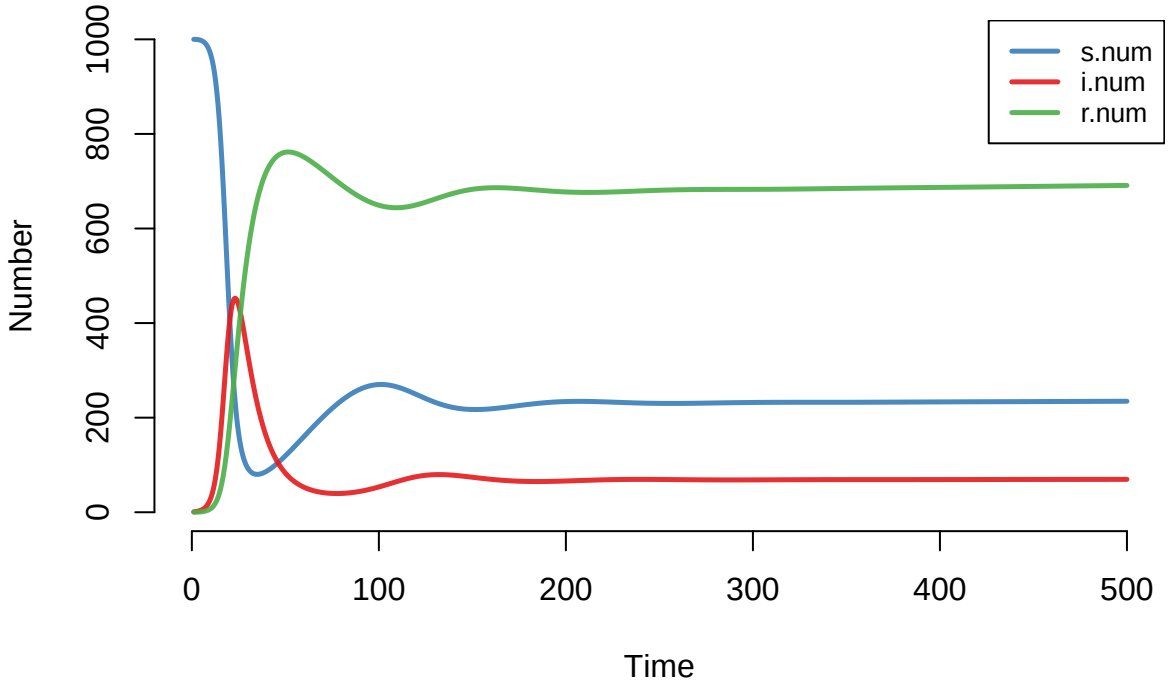

```
par(mfrow = c(1, 1))
comp_plot(SISmod, at = 100, digits = 1)
```



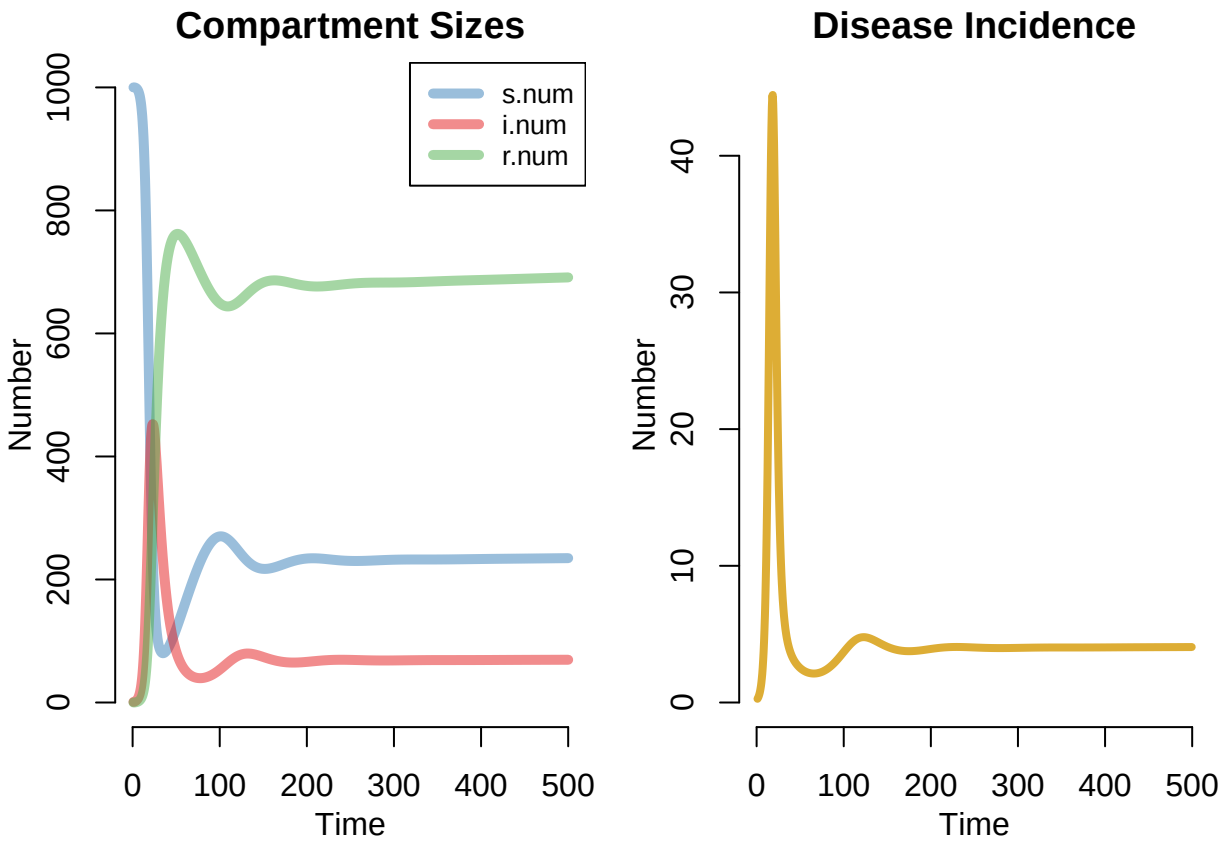
Demography:

We can also incorporate Demography which will be done by including the concepts of death rates and death rates where b is the birth rate, and mu is the death rate. This system is explained by the following ODEs: dS/dt = -lambda$S$ + bN -mu_sS dI/dt = $lambda$S - nu$I$ -mu_iS dR/dt = nu$I$ -mu_rR

```
param <- param.dcm(inf.prob = 0.33, act.rate = 1.5, rec.rate = 1/10,
                   b.rate = 1/95, ds.rate = 1/100, di.rate = 1/60, dr.rate = 1/100) #b.rate= birth rate
init <- init.dcm(s.num = 1000, i.num = 1, r.num = 0) #adding in r compartment
control <- control.dcm(type = "SIR", nsteps = 500, dt = 0.5) # dt is here so we can obtain data within
SIRmod_2 <- dcm(param, init, control)
plot(SIRmod_2)
```




```
#and to plot it, first for compartment numbers and then for prevalence:

par(mar = c(3.2, 3, 2, 1), mgp = c(2, 1, 0), mfrow = c(1, 2)) # par() used to set graphical parameters,
# the par() is being used for the first plot
plot(SIRmod_2, popfrac = FALSE, alpha = 0.5,
     lwd = 5, main = "Compartment Sizes") #popfrac= FALSE plots the compartment size rather than the pr

plot(SIRmod_2, y = "si.flow", lwd = 4, col = "goldenrod",main = "Disease Incidence", legend = "n")
```



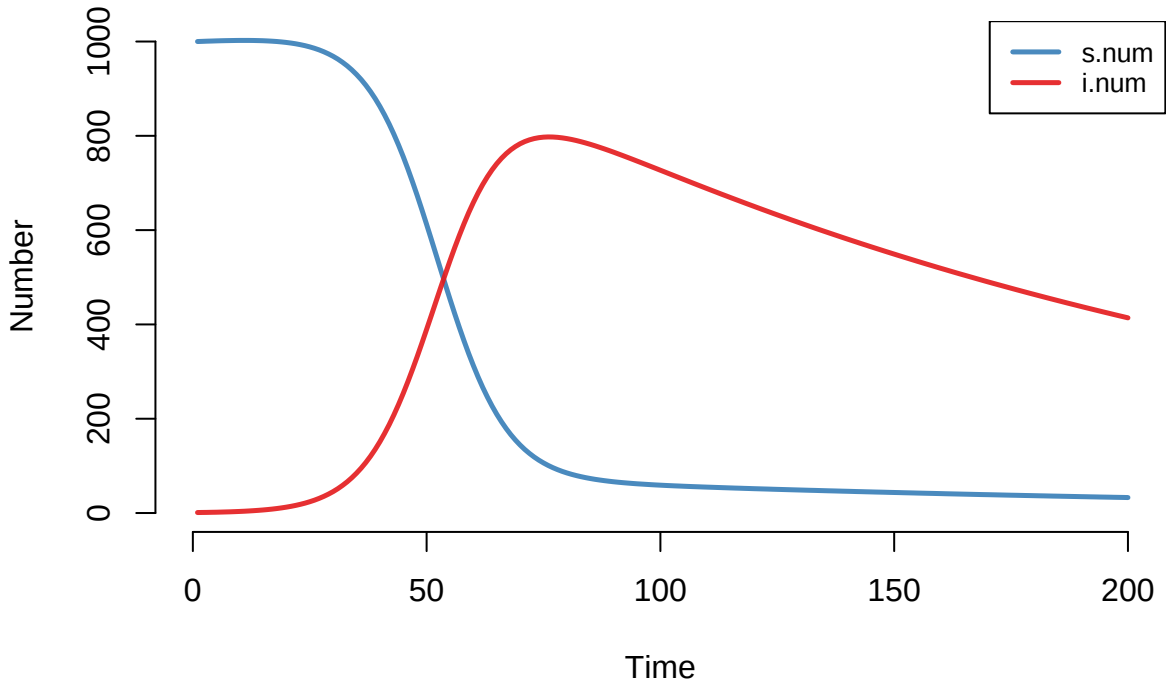
And SI with demography:

```
param <- param.dcm(inf.prob = 0.3, act.rate = 0.5, b.rate = 1/95, ds.rate = 1/100, di.rate = 1/60)

init <- init.dcm(s.num = 1000, i.num = 1) #
control <- control.dcm(type = "SI", nsteps = 200)

SImod_2 <- dcm(param, init, control)
SImod_2
```

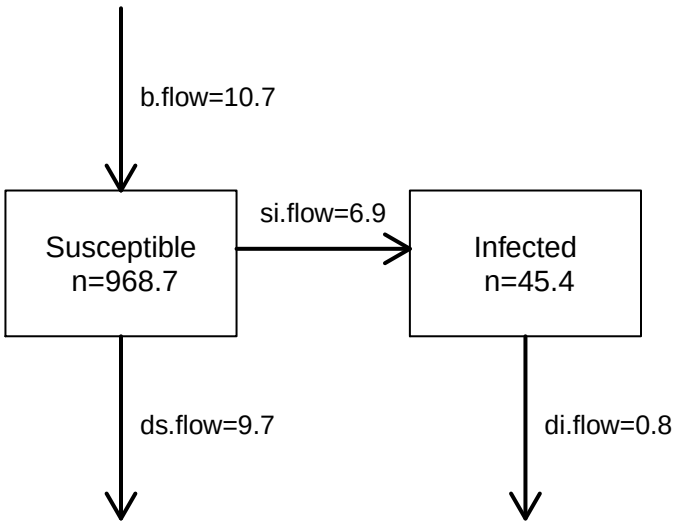
```
## EpiModel Simulation
## =======================
## Model class: dcm
##
## Simulation Summary
## ----------------------
## Model type: SI
## No. runs: 1
## No. time steps: 200
## No. groups: 1
##
## Model Parameters
## ----------------------
## inf.prob = 0.3
## act.rate = 0.5
## b.rate = 0.01052632
## ds.rate = 0.01
```

```
## di.rate = 0.01666667
##
## Model Output
## -----------------------
## Variables: s.num i.num si.flow b.flow ds.flow di.flow num
```

```
plot(SImod_2)
```




```
par(mfrow = c(1, 1))
comp_plot(SImod_2, at = 30, digits = 1)
```



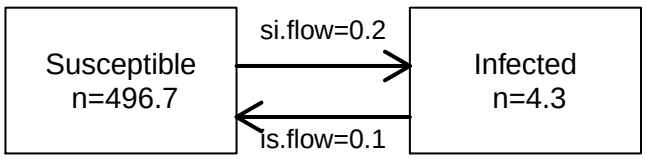
Sensitivity analyses: We can run a sensitivity analyses to study how the model varies undera a variety of parameters. To do this, the parameter of interest must be entered in the form of a vector. This will enable the `dcm` function to run a set of models

```
param <- param.dcm(inf.prob = 0.2, act.rate = seq(0.25, 0.5, 0.05), rec.rate = 0.02)#vector here, enabl
init <- init.dcm(s.num = 500, i.num = 1)
control <- control.dcm(type = "SIS", nsteps = 250)
SISmod_2 <- dcm(param, init, control)
SISmod_2 # so we can examine the model
```

```
## EpiModel Simulation
## =======================
## Model class: dcm
##
## Simulation Summary
## ----------------------
## Model type: SIS
## No. runs: 6
## No. time steps: 250
## No. groups: 1
##
## Model Parameters
## ----------------------
## inf.prob = 0.2
## act.rate = 0.25 0.3 0.35 0.4 0.45 0.5
## rec.rate = 0.02
##
```

```
## Model Output
## -----------------------
## Variables: s.num i.num si.flow is.flow num
```

```
comp_plot(SISmod_2, at = 50, digits = 1) # a visualization of the flow diagram
```



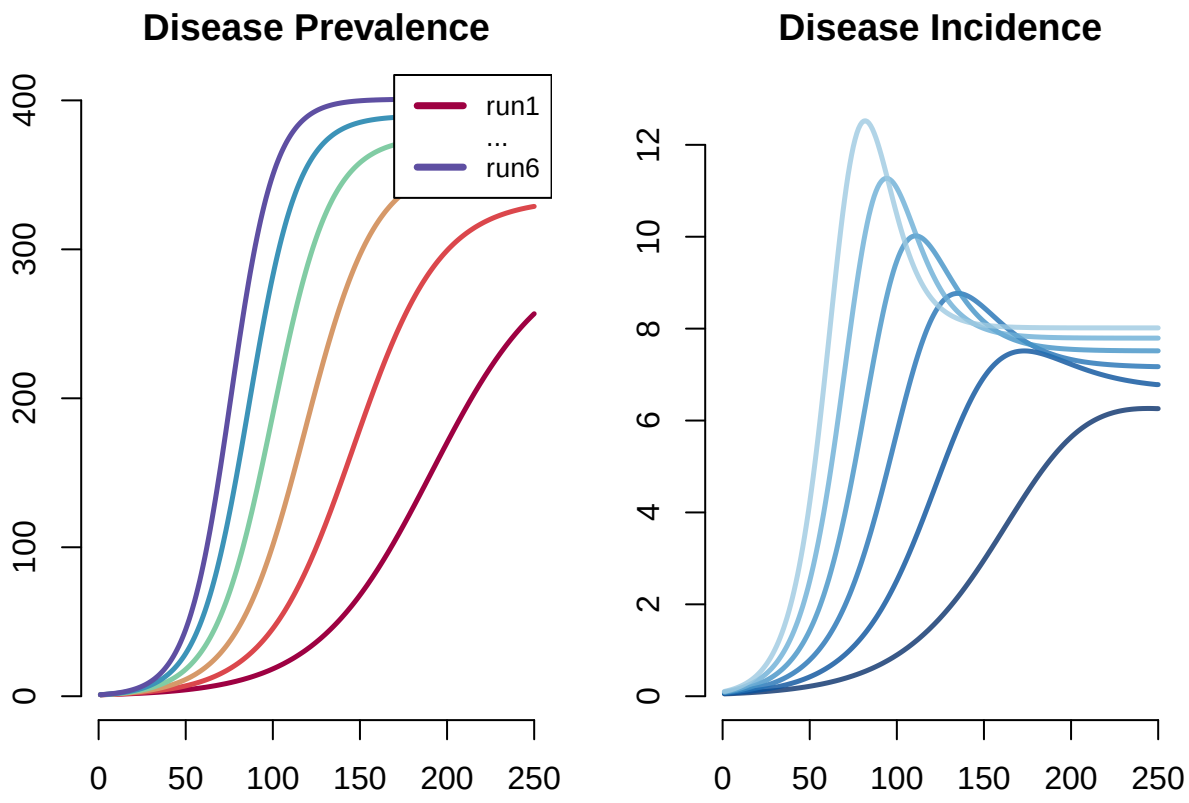
To extract the data from the model for graphing a specific run we use `as.data.frame`

```
z <- head(as.data.frame(SISmod_2, run=4))
z
```

```
##   time    s.num    i.num    si.flow    is.flow num
## 1    1 500.0000 1.000000 0.08227231 0.02061048 501
## 2    2 499.9383 1.061662 0.08733382 0.02188125 501
## 3    3 499.8729 1.127114 0.09270503 0.02323013 501
## 4    4 499.8034 1.196589 0.09840467 0.02466189 501
## 5    5 499.7297 1.270332 0.10445258 0.02618158 501
## 6    6 499.6514 1.348603 0.11086977 0.02779457 501
```

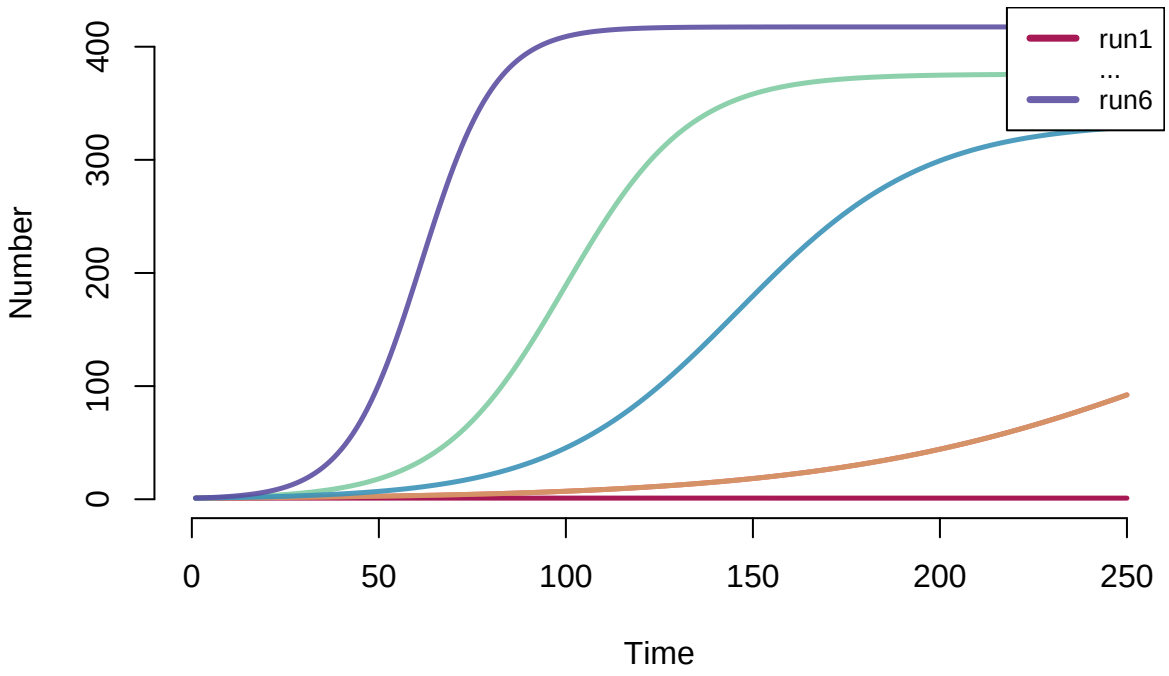
To plot the data, showing prevalance and incidence across all runs:

```
par(mfrow = c(1,2), mar = c(3.2,3,2.5,1))
plot(SISmod_2, alpha = 1, main = "Disease Prevalence")
plot(SISmod_2, y = "si.flow", col = "Blues", alpha = 0.8, main = "Disease Incidence")
```



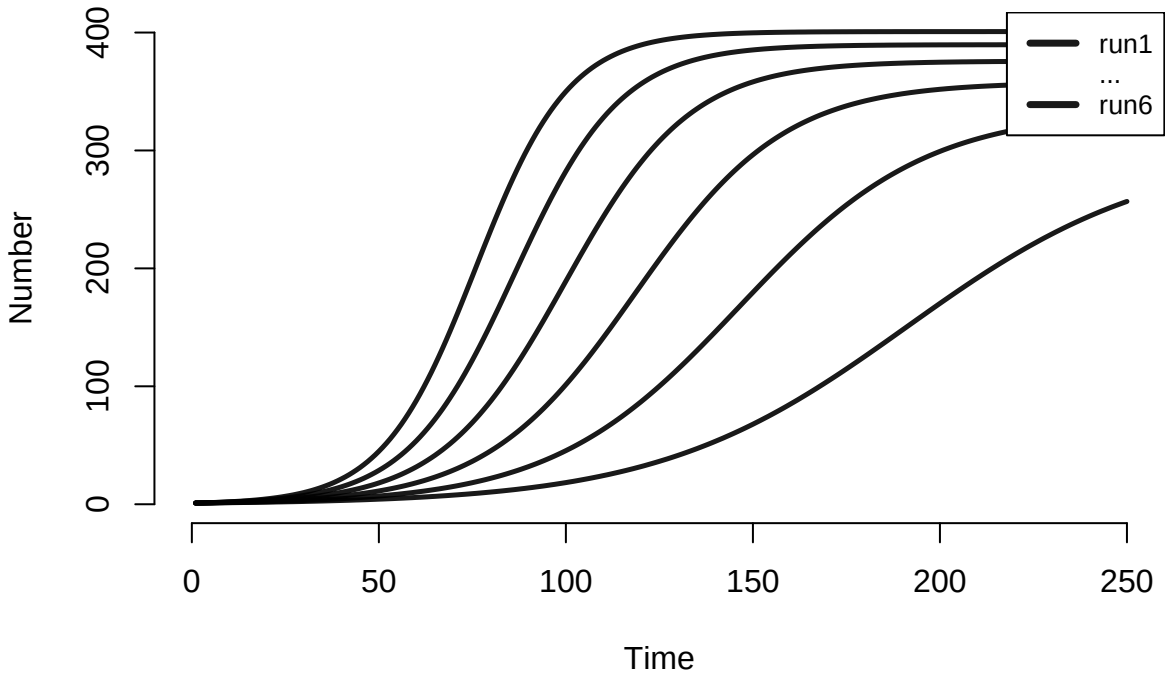
We can also have variation in more than one parameters but to do so the parameters must be vectors of equal length.

```
act.rates <- c(0.2, 0.2, 0.4, 0.4, 0.6, 0.6)
inf.probs <- c(0.1, 0.2, 0.1, 0.2, 0.1, 0.2)
param <- param.dcm(inf.prob = inf.probs, act.rate = act.rates,
                   rec.rate = 0.02)
init <- init.dcm(s.num = 500, i.num = 1)
control <- control.dcm(type = "SIS", nsteps = 250)
SISmod_3 <- dcm(param, init, control)
plot(SISmod_3)
```

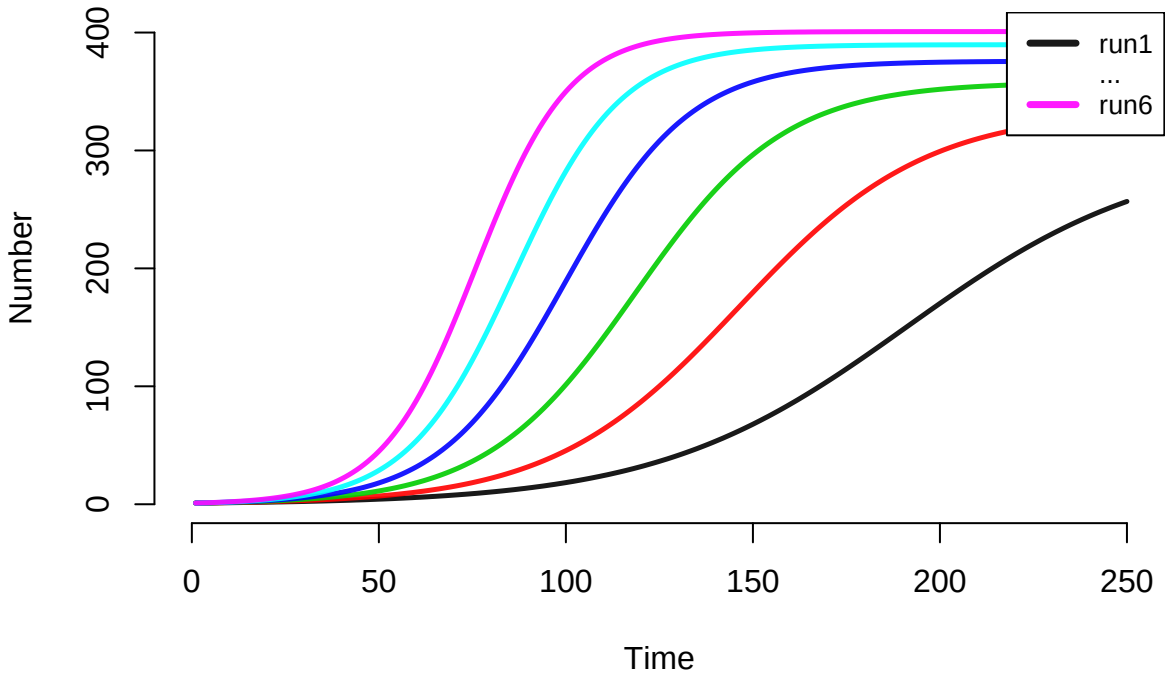


And we can further alter colors as EpiModel uses `RColorBrewer`. The default palette is used in a prevalence plot, but we can use any palette in the `display.brewer.all()` function:

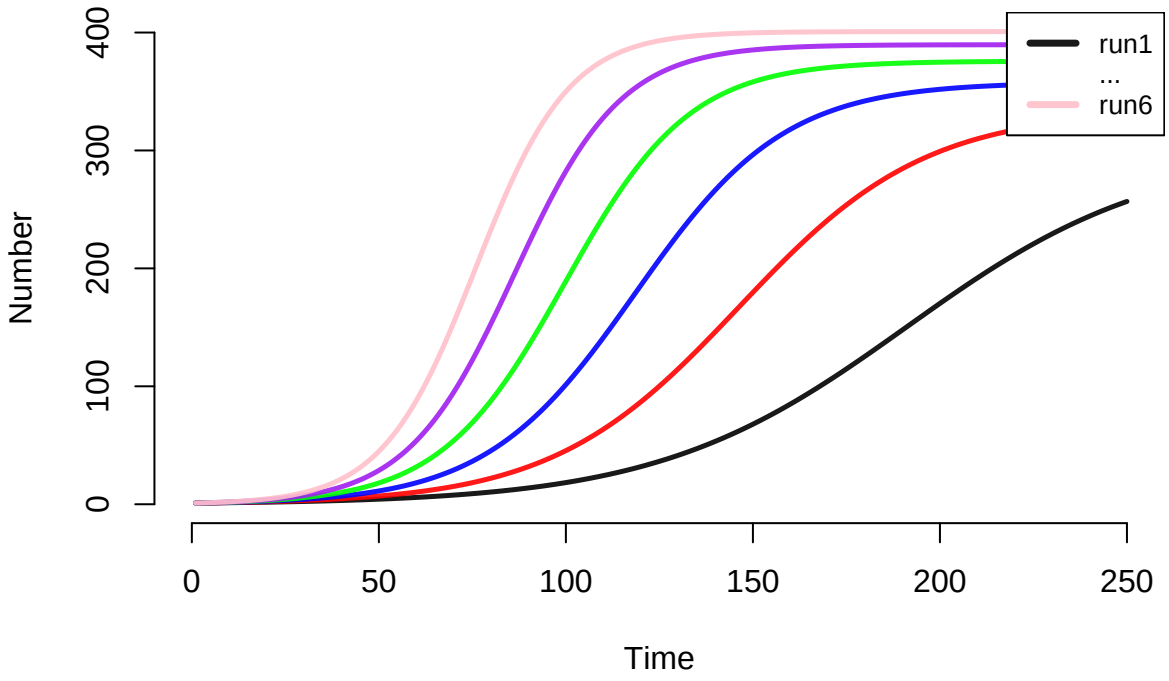
```
plot(SISmod_2, col = "black")
```



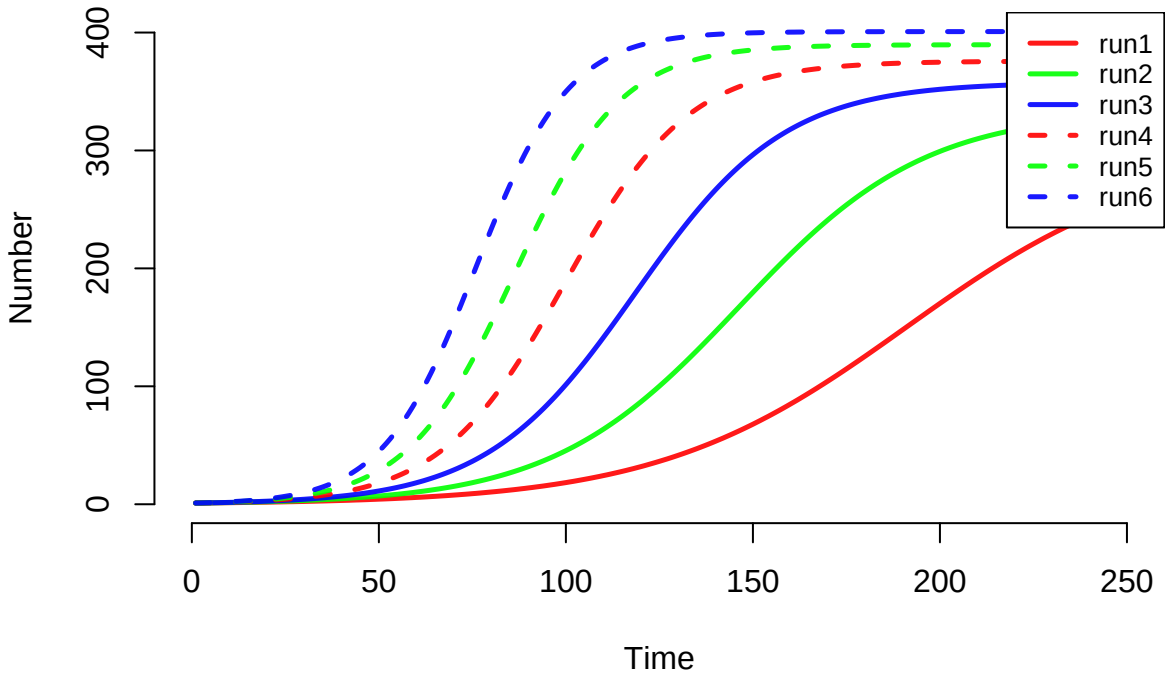

```
plot(SISmod_2, col = 1:6)
```




```
plot(SISmod_2, col = c("black", "red", "blue", "green", "purple", "pink"))
```




```
plot(SISmod_2, col = rainbow(3), lty = rep(1:2, each = 3), legend = "full") #enables us to set run-spec
```



Two-group models In EpiModel thus far we have focused on creating one-group models with random mixing implied. It is also possible to create two-group, or bipartite, models. The assumptions of these models is there is no contact within groups and only cross-contact–therefore mixing is heterogeneous. This makes such a bipartite model useful for modeling heterosexual relationships.

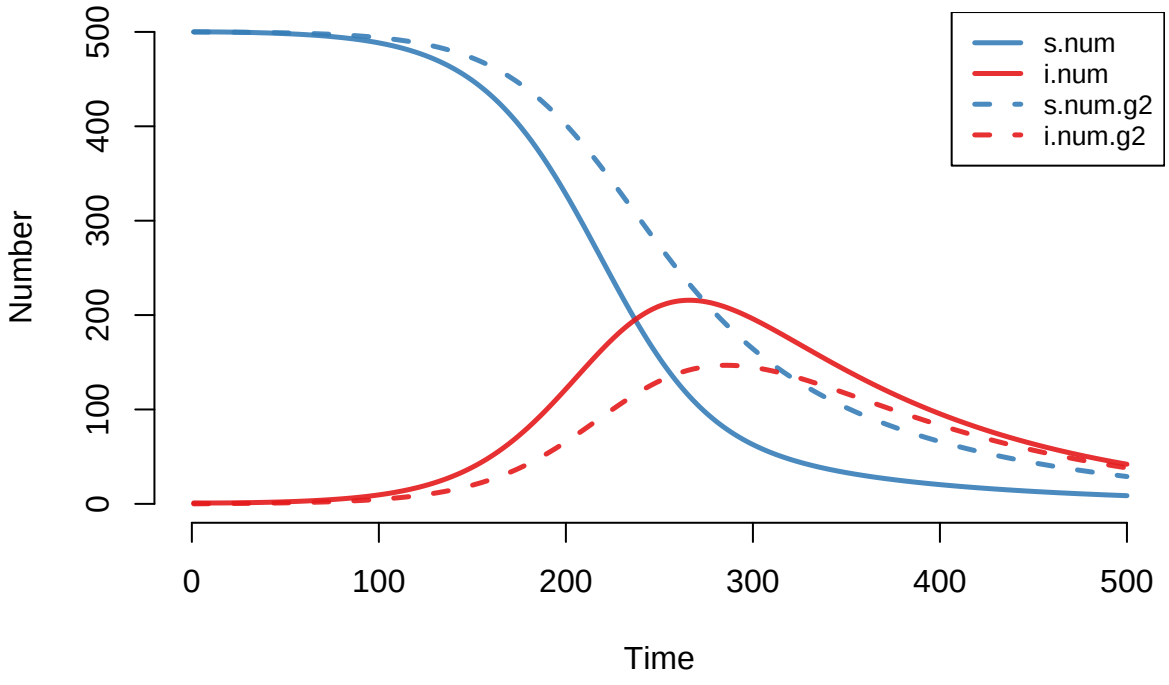
SI bipartite model with demography: We will now have four compartments (two groups*two disease states). Without going into the mathematics and reasoning behind it. In this example group 1 is female, and group 2 is male as will be seen. This is because the birth rate is only the function of one group. Furthermore, in order to make sure that heterogeneity occurs, we must include a `balance=` argument in the `param()`. By doing so we specify `act.rate` for one group and state that one group has a controlling rate.

```
param <- param.dcm(inf.prob=0.4, inf.prob.g2 = 0.1, act.rate=0.25, balance= "g1",
                   b.rate=1/100, b.rate.g2 = NA, ds.rate=1/100, ds.rate.g2=1/100,
                   di.rate =1/50, di.rate.g2=1/50)

# here we establish the birth and death rates for the group groups. Incorporating a balance to g1 enabl
init <- init.dcm(s.num = 500, i.num = 1, s.num.g2 = 500, i.num.g2 = 0) #same principal here
control <- control.dcm(type = "SI", nsteps = 500)
SImod_3 <- dcm(param, init, control)
SImod_3
```

```
## EpiModel Simulation
## =======================
## Model class: dcm
##
## Simulation Summary
## -----------------------
```

```
## Model type: SI
## No. runs: 1
## No. time steps: 500
## No. groups: 2
##
## Model Parameters
## ------------------------
## inf.prob = 0.4
## act.rate = 0.25
## b.rate = 0.01
## ds.rate = 0.01
## di.rate = 0.02
## inf.prob.g2 = 0.1
## b.rate.g2 = NA
## ds.rate.g2 = 0.01
## di.rate.g2 = 0.02
## balance = g1
##
## Model Output
## ------------------------
## Variables: s.num i.num s.num.g2 i.num.g2 si.flow b.flow
## ds.flow di.flow si.flow.g2 b.flow.g2 ds.flow.g2 di.flow.g2
## num num.g2
```

```
plot(SImod_3)
```



ICMs Basic ICMS, or stochastic individual contact models. ICMs are designed to be agent-based simulation

do DCMs under `dcm()`. We will now be using the function `icm()`. Main differences between the two modeling types: 1) ICMs have parameters that are random draws and explained by rates/probabilities ie: normal, Poisson distribution 2) ICMs use discrete time rather than continuous time 3)ICMs simulate spread of disease over individuals rather than DCMs which treat population as a whole (so the individuals are not identifiable).

SI ICM:

```
param <- param.icm(inf.prob = 0.2, act.rate = 0.25)# note the function is now param.icm
init <- init.icm(s.num = 500, i.num = 1)
control <- control.icm(type = "SI", nsims = 10, nsteps = 300)
SImod <- icm(param, init, control)
SImod
```

```
## EpiModel Simulation
## =======================
## Model class: icm
##
## Simulation Summary
## ----------------------
## Model type: SI
## No. simulations: 10
## No. time steps: 300
## No. groups: 1
##
## Model Parameters
## ----------------------
## inf.prob = 0.2
## act.rate = 0.25
##
## Model Output
## ----------------------
## Variables: s.num i.num num si.flow
```

```
summary(SImod, at = 125) # to request summary of individual timestep 125
```

```
## EpiModel Summary
## =======================
## Model class: icm
##
## Simulation Details
## ----------------------
## Model type: SI
## No. simulations: 10
## No. time steps: 300
## No. groups: 1
##
## Model Statistics
## ------------------------------
## Time: 125
## ------------------------------
##            mean      sd    pct
## Suscept.  335.0  85.267  0.669
## Infect.   166.0  85.267  0.331
## Total     501.0   0.000  1.000
## S -> I      5.1   1.853     NA
## ------------------------------
```

```
head(as.data.frame(SImod, out= "mean"))
```

```
##   time s.num i.num num si.flow
## 1    1 500.0   1.0 501     0.0
## 2    2 500.0   1.0 501     0.0
## 3    3 500.0   1.0 501     0.0
## 4    4 500.0   1.0 501     0.0
## 5    5 499.9   1.1 501     0.1
## 6    6 499.9   1.1 501     0.0
```

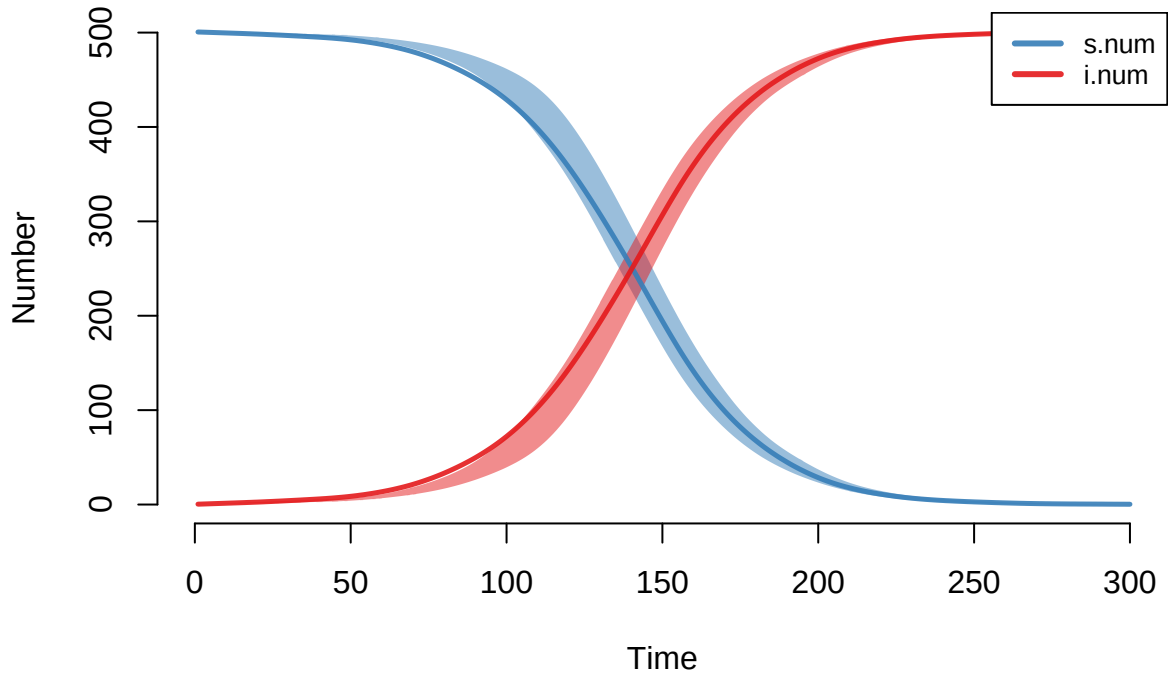
```
tail(as.data.frame(SImod, out = "vals", sim = 1))
```

```
##     sim time s.num i.num num si.flow
## 295   1  295     0   501 501       0
## 296   1  296     0   501 501       0
## 297   1  297     0   501 501       0
## 298   1  298     0   501 501       0
## 299   1  299     0   501 501       0
## 300   1  300     0   501 501       0
```

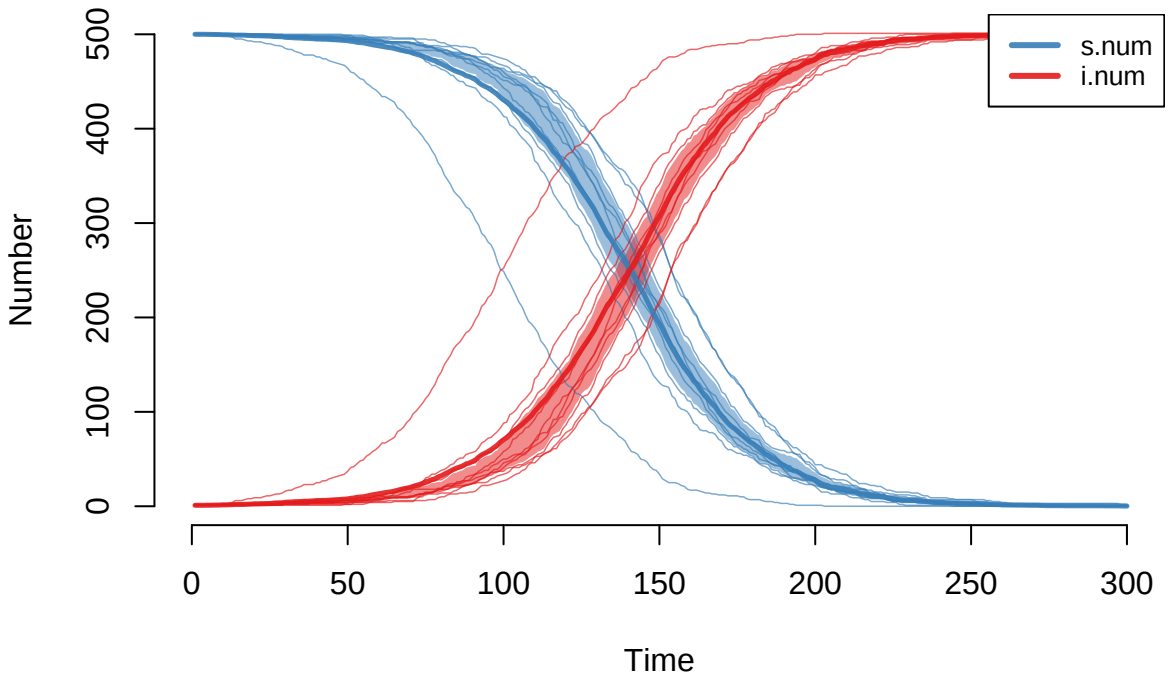
```
# can be used to find summary stats of interest and are time-specific
```

For plotting we can alter the arguments within `plot.icm`:

```
plot(SImod) # standard plot
```




```
plot(SImod, sim.lines = TRUE, mean.smooth = FALSE, qnts.smooth = FALSE)
```



Comparing SIR for `dcm` and `icm`

```
#dcm
param <- param.dcm(inf.prob = 0.3, act.rate = 0.92, rec.rate = 1/50,
                   b.rate = 1/50, ds.rate = 1/60, di.rate = 1/40, dr.rate = 1/60)
init <- init.dcm(s.num = 800, i.num = 200, r.num = 0)
control <- control.dcm(type = "SIR", nsteps = 300)
mD <- dcm(param, init, control)
# icm
param <- param.icm(inf.prob = 0.3, act.rate = 0.92, rec.rate = 1/50,
                   b.rate = 1/50, ds.rate = 1/60, di.rate = 1/40,
                   dr.rate = 1/60)
init <- init.icm(s.num = 800, i.num = 200, r.num = 0)
control <- control.icm(type = "SIR", nsteps = 300, nsims = 10)
mI <- icm(param, init, control)
```

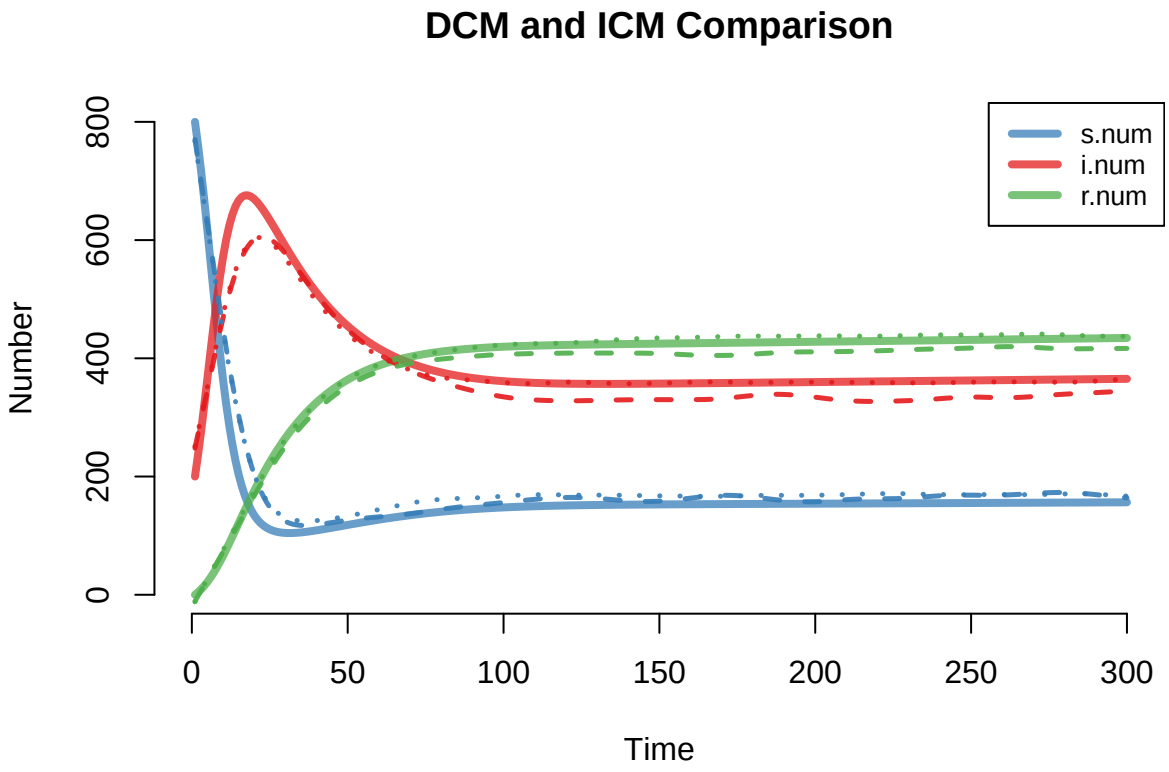
In a third model, an `icm` we will remove stochastic birth and death rates (generally random draws from binomial distributions)

```
control <- control.icm(type = "SIR", nsteps = 300, nsims = 10,
                       b.rand = FALSE, d.rand = FALSE) #this should cause less variability
mI_2 <- icm(param, init, control)
```

Plotting these three functions together, using the `add` argument. Note that the means for the models seem to be relatively the same

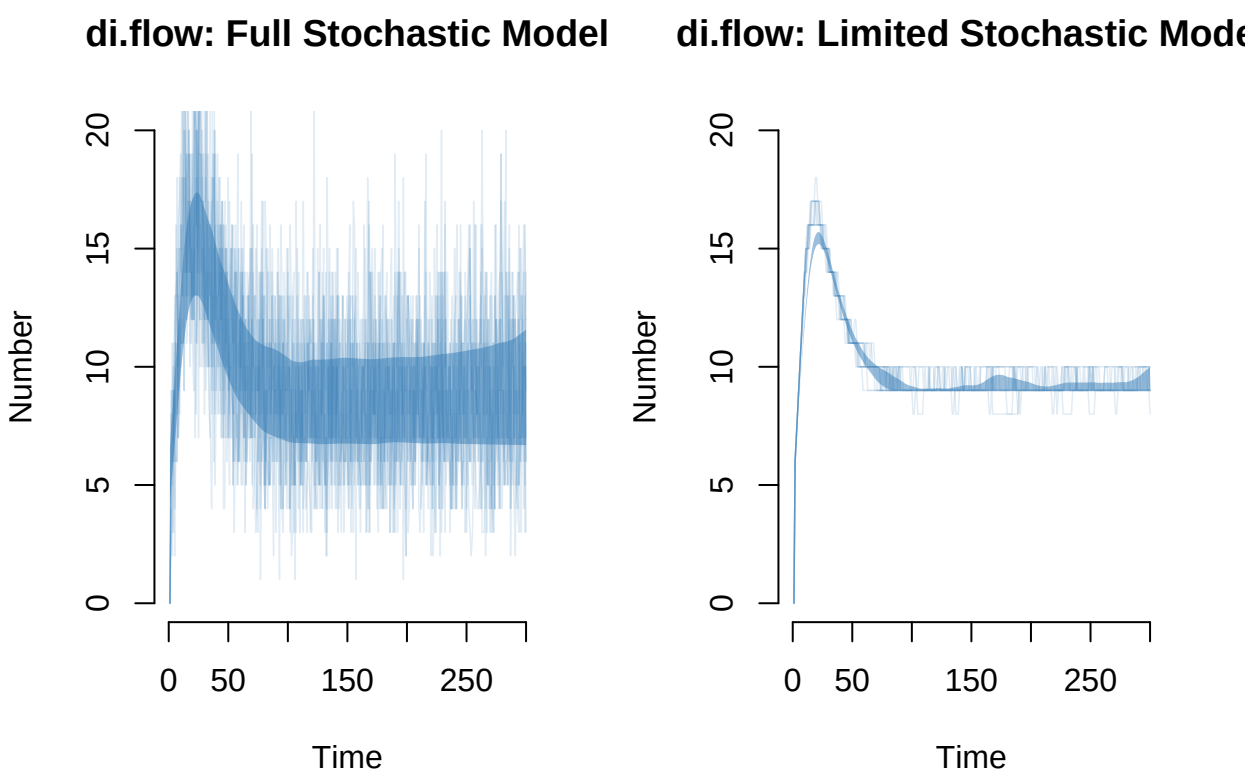
```
plot(mD, alpha = 0.75, lwd = 4, main = "DCM and ICM Comparison")
plot(mI, qnts = FALSE, sim.lines = FALSE, add = TRUE, mean.lty = 2, legend = FALSE)
```

```
plot(mI_2, qnts = FALSE, sim.lines = FALSE, add = TRUE, mean.lty = 3, legend = FALSE)
```



We can also compare the variation between the two icms, one that has stochastic birth/death rates and one that doesn't. Note the much wider range for the model that is fully stochastic:

```
par(mfrow = c(1,2))
plot(mI, y = "di.flow", mean.line = FALSE,
     sim.lines = TRUE, sim.alpha = 0.15,
     main = "di.flow: Full Stochastic Model",
     ylim= c(0,20))

plot(mI_2, y = "di.flow", mean.line = FALSE,
     sim.lines = TRUE, sim.alpha = 0.15,
     main = "di.flow: Limited Stochastic Model",
     ylim= c(0,20))
```

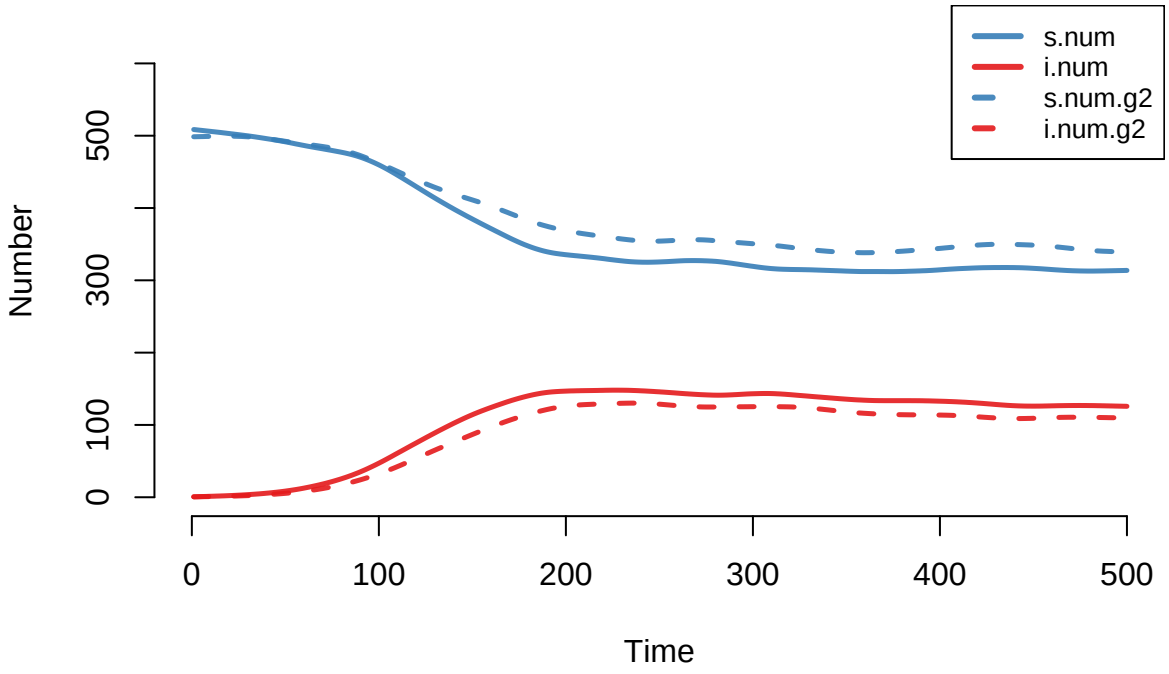



```
#add limits ylim = c(0,20)
```

It's cool to see how simply removing stochasticity in b/d rates would drastically reduce overall variability.

SIS bipartite with demography

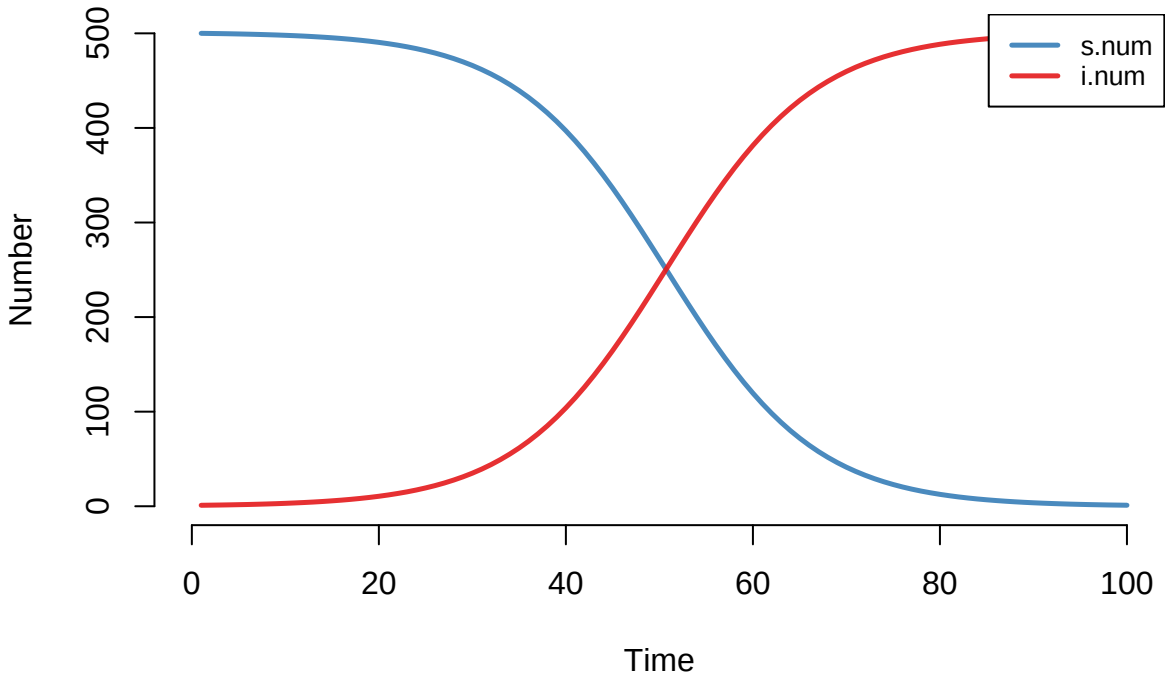
```
param <- param.icm(inf.prob = 0.3, inf.prob.g2 = 0.1, act.rate = 0.5, balance = "g1",
                   rec.rate = 1/25, rec.rate.g2 = 1/50, b.rate = 1/100, b.rate.g2 = NA,
                   ds.rate = 1/100, ds.rate.g2 = 1/100, di.rate = 1/90, di.rate.g2 = 1/90)
init <- init.icm(s.num = 500, i.num = 1, s.num.g2 = 500, i.num.g2 = 1)
control <- control.icm(type = "SIS", nsteps = 500, nsims = 10)


sisI <- icm(param, init, control)
plot(sisI)
```



note that only the means will be printed for a bipartite ICM.

Part II- Making a model Let's take the SI model and add `print.mod=TRUE` as an argument in `control()`. Solved using the `deSolve` package. This will enable us to look at the model structure in the console:

```
param <- param.dcm(inf.prob = 0.5, act.rate = 0.25)
init <- init.dcm(s.num = 500, i.num = 1)
control <- control.dcm(type = "SI", nsteps = 100)
mod <- dcm(param, init, control)
plot(mod)
```



now adding that extra argument:

```
control <- control.dcm(type = "SI", nsteps = 100, print.mod = TRUE)
mod <- dcm(param, init, control)
```

```
## function (t, t0, parms)
## {
##     with(as.list(c(t0, parms)), {
##         num <- s.num + i.num
##         lambda <- inf.prob * act.rate * i.num/num
##         if (!is.null(parms$inter.eff) && t >= inter.start) {
##             lambda <- lambda * (1 - inter.eff)
##         }
##         si.flow <- lambda * s.num
##         dS <- -si.flow
##         dI <- si.flow
##         list(c(dS, dI, si.flow), num = num)
##     })
## }
## <environment: namespace:EpiModel>
```

The top two lines of the model wrap input and output into a list. The calculation of derivatives are functions of parameters. There is also a modifier on the lambda to cause an "intervention" if specific parameters are specified in the model. The following line will also print the same model in the console: `print(mod_SI_1g_cl)` and all other models can be viewed as well `print(mod_SIR_1g_op)`

To write a model function we must: 1) include the overall function structure including the line that puts the input and output into a list 2) dynamic calculations for things such as lamba, total sum `num`, values for

s.num, i.num, and composite statistics: `prevalence <- i.num/num 3) ODEs, solved by`deSolve"' go here. Names must be consistent with output list. 4) the output list. derivatives first in same order that they are entered in initial conditions 5) function should be given a name that is relevant

In defining the paramters and control settings for model in `param.dcm()`, `init,dcm()`, and `control.dcm()` 1) enter parameters is you would in a normal model 2) ******states must be in same order as ODEs in function output. Note that flows need to end with `.flow` 3) controls are entered similarly.

An example: creating an SEIR model (examples in tutorial). E represents an exposed compartment, and we will include the parameter R_0. Furthermore, this will be a closed population with only deaths from Ebola. other assumptions are the the dead to not transmit disease, contact rate is same over all of the infected phase.

Thus we will add `R0` the initial reproductive numner, `e.dur` as duration of exposed state, `i.dur` as duration of infectious state, and `cfr` as case fatality rate. Via R_0 we can find a `lambda` . 4 ODEs will be necessary:

```
SEIR <- function(t, t0, parms) {
  with(as.list(c(t0, parms)), {

    # Population size
    num <- s.num + e.num + i.num + r.num

    # Effective contact rate and FOI from a rearrangement of Beta * c * D
    ce <- R0 / i.dur
    lambda <- ce * i.num/num

    dS <- -lambda*s.num
    dE <- lambda*s.num - (1/e.dur)*e.num
    dI <- (1/e.dur)*e.num - (1 - cfr)*(1/i.dur)*i.num - cfr*(1/i.dur)*i.num
    dR <- (1 - cfr)*(1/i.dur)*i.num

    # Compartments and flows are part of the derivative vector
    # Other calculations to be output are outside the vector, but within the containing list
    list(c(dS, dE, dI, dR,
           se.flow = lambda * s.num,
           ei.flow = (1/e.dur) * e.num,
           ir.flow = (1 - cfr)*(1/i.dur) * i.num,
           d.flow = cfr*(1/i.dur)*i.num),
         num = num,
         i.prev = i.num / num,
         ei.prev = (e.num + i.num)/num)
  })
}
```

now we will run the model:

```
param <- param.dcm(R0 = 1.9, e.dur = 10, i.dur = 14, cfr =  0.9)
init <- init.dcm(s.num = 1e6, e.num = 10, i.num = 0, r.num = 0,
                 se.flow = 0, ei.flow = 0, ir.flow = 0, d.flow = 0)
control <- control.dcm(nsteps = 500, dt = 1, new.mod = SEIR)
mod <- dcm(param, init, control)
mod
```

```
## EpiModel Simulation
## =======================
## Model class: dcm
##
## Simulation Summary
```

```
## ----------------------
## No. runs: 1
## No. time steps: 500
##
## Model Parameters
## ----------------------
## R0 = 1.9
## e.dur = 10
## i.dur = 14
## cfr = 0.9
##
## Model Output
## ----------------------
## Variables: s.num e.num i.num r.num se.flow ei.flow ir.flow
## d.flow num i.prev ei.prev
```

And Plot:

```
par(mfrow = c(1, 2))
plot(mod, y = "i.num", main = "Prevalence")
plot(mod, y = "se.flow",  main = "Incidence")
```